CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Facultad de Informática

TRABAJO FIN DE GRADO

Detección de marcas para realidad aumentada sobre dispositivos moviles

Autor: Santiago Valverde García

Director: Luis Baumela Molina

MADRID, ENERO DE 2015

ÍNDICE

1. RESUMEN

Este documento es una guía para el desarrollo de una aplicación para dispositivos móviles en Android. Dicha aplicación combina las técnicas de visión por computador para calibrar la cámara del dispositivo y localizar un elemento en el espacio en base a esos los parámetros calculados en la calibración.

El diseño de la aplicación incluye las decisiones sobre la forma en que se reciben los inputs de la aplicación, que patrones se utilizan en la calibración y en la localización y como se muestran los resultados finales al usuario. También incluye un diagrama de flujo de información que representa el tránsito de esta entre los diferentes módulos.

La implementación comienza con la configuración de un entorno para desarrollar aplicaciones con parte nativa en Android, después comenta el código de la aplicación paso por paso incluyendo comentarios sobre los archivos adicionales necesarios para la compilación y finalmente detalla los archivos dedicados a la interfaz.

Los experimentos incluyen una breve descripción sobre cómo interpretar los resultados seguidos de una serie de imágenes tomadas de la aplicación con diferentes localizaciones del patrón. En la entrega se incluye también un video.

En el capítulo de resultados y conclusiones podemos encontrar observaciones sobre el desarrollo de la práctica, opiniones sobre su utilidad, y posibles mejoras.

This document is a guide that describes the development of and application for mobile devices in Android OS. The application combines computer vision techniques to calibrate the device camera and locate an element in the real world based on the parameters of the calibration

The design of the application includes the decisions over the way that the application receives its input data, the patterns used in the calibration and localization and how the results are shown to the user. It also includes a flow chart that describes how the information travels along the application modules.

The development begins with the steps necessary to configure the environment to develop native Android applications, then it explains the code step by step, including commentaries on the additional files necessary to build the application and details the files of the user interface.

The experiments chapter explains the way the results are shown in the experiments before showing samples of different pattern localizations. There is also a video attached.

In the conclusions chapter we can find observations on the development of the TFG, opinions about its usefulness, and possibilities of improvement in the future.

2. INTRODUCCIÓN

2.1 Planteamiento

El objetivo del trabajo de fin de grado (TFG) es el desarrollo de una aplicación en Android de una aplicación capaz de reconocer marcas situadas en un entorno real a través de la cámara del móvil.

Esta aplicación será capaz de 3 cosas:

1. Calibrar la cámara mediante un set de imágenes previamente capturadas
2. Detectar las marcas en el entorno a través de la cámara
3. Proveer al usuario información sobre la posición relativa de las marcas, de forma que esta información pueda ser usada posteriormente para proyectar imágenes.

Para llevar a cabo el proyecto desarrollaré la aplicación en el entorno eclipse con el plugin CDT para manejar las partes nativas del programa, usare las herramientas del NDK de Android y la librería de OpenCV para el calibrado y procesado de imágenes.

El TFG incluirá además una guía para facilitar el uso tanto de la aplicación como de los componentes usados para desarrollarla. Esta guía comprenderá:

1. Instalación de eclipse con el plugin CDT
2. Instalación del SDK de Android
3. Instalación del NDK de Android
4. Instalación de OpenCV
5. Configuración de un proyecto Android usando las anteriores herramientas
6. Manual de usuario de la aplicación

Además, dado que este trabajo puede que sea propuesto como practica a futuros alumnos, la memoria está dirigida a estos e intenta informar desde un punto de vista cercano. Por esta razón toda la memoria está planteada como una guía para aquellos que estén interesados en hacer un proyecto similar. Para ello usa un tono cercano y acompaña más que guía a través del desarrollo.

2.2. Interés

El interés general de la aplicación se puede dividir en 2 partes:

- Su utilidad como herramienta de desarrollo: El primer objetivo de la aplicación es que se use como base para aplicaciones de realidad aumentada. Su utilidad va desde aplicaciones de entretenimiento, como los “Invizimals” de las plataformas

de entretenimiento de Sony; hasta proyección de datos y/o descripciones sobre elementos reales.

- Su utilidad como elemento docente: El segundo objetivo de la aplicación es ser usada como propuesta de práctica docente. Por este motivo es importante incluir un manual de usuario enfocado a facilitar la comprensión de la práctica a los alumnos y profesores a quien pueda interesar.

Finalmente este trabajo tiene un interés particular para mí. Este trabajo me permite combinar varias áreas en las que estoy interesado como son la visión por computador y el desarrollo de aplicaciones para Android. Además, el desarrollo nativo en Android es una característica muy interesante y este trabajo me permite ahondar en ella.

3. ESTADO DEL ARTE

En este capítulo trataré las herramientas y el entorno en los que desarrollaré la aplicación explicando en qué consiste cada uno de ellos y qué papel desempeñan en el desarrollo del trabajo.

3.1. Entorno de desarrollo: Eclipse

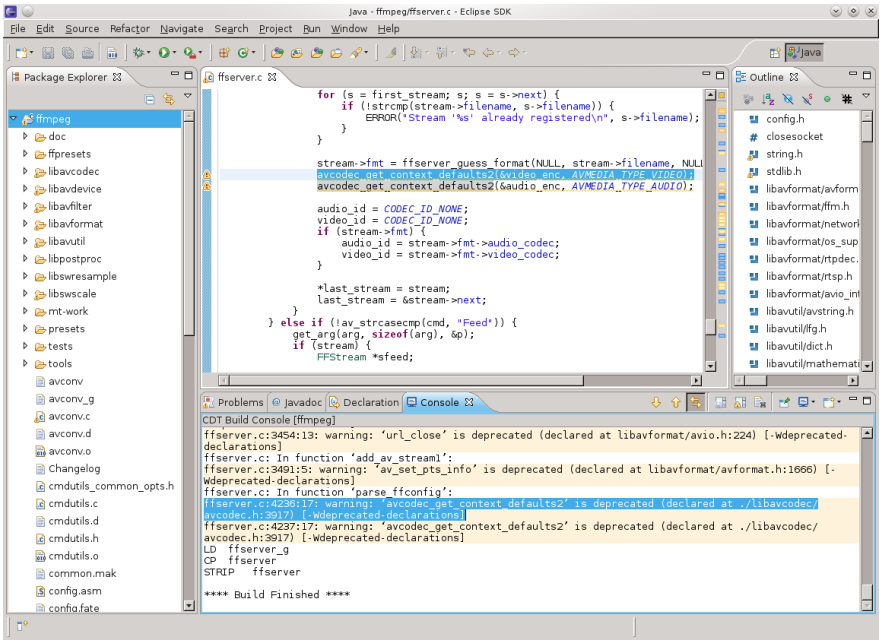
La Wikipedia define un entorno de desarrollo como:

Un entorno de desarrollo integrado, llamado también IDE (sigla en inglés de integrated development environment), es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación; es decir, que consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IIDDEE pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, PHP, Python, Java, C#, Delphi, Visual Basic, Gambas, etc. En algunos lenguajes, un IDE puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto, como es el caso de Smalltalk u Objective-C.

Existen numerosos entornos de desarrollo, como pueden ser: Eclipse, Netbeans, Visual Studio o Codeblocks.



Ejemplo de interfaz del IDE Eclipse

Eclipse es el entorno de desarrollo que he elegido para llevar a cabo el trabajo. Es un entorno gratuito con múltiples funcionalidades extensibles a través de plugins. Incorpora herramientas para debuguear, compilar, editar código en varios lenguajes y previsualizar errores.

Personalmente es el entorno en el que he trabajado siempre que he programado en java y en las eventuales ocasiones en que he desarrollado para Android. Aunque es cierto que no es el entorno ideal, facilita la integración de Java, C++, Android y OpenCV enormemente por lo que, añadido a la familiaridad ya adquirida, decidí decantarme por este entorno.

3.2. CDT (C/C++ Development Tooling) Plugin

Un plugin se define como una aplicación desarrollada expresamente para complementar o extender otra aplicación ya creada de forma que añada utilidades o herramientas. Algunos tipos de aplicaciones que permiten el uso de plugins son los navegadores web, los reproductores de audio y los IDEs anteriormente comentados.

Existen plugins de uso muy común, como el adblock, que es un plugin para navegador web que nos bloquea las ventanas emergentes de publicidad y otros elementos; los plugins de extensión de formato, que permiten a un reproductor reproducir formatos originalmente no soportados en un reproductor de audio o video; o el adobe flash

player, quizá el más famoso, que nos permite reproducir archivos sfw desarrollados con el adobe flash en navegadores web.

El plugin CDT se integra en eclipse y permite a eclipse manejar proyectos en lenguaje C o C++. Entre las utilidades que añade a eclipse se encuentran el reconocimiento de la sintaxis de C y C++ para previsualizar errores de compilación y la inclusión de menús en la interfaz de eclipse que permiten configurar proyectos C/C++ para usar compiladores alternativos y librerías externas.

Puesto que buena parte de la aplicación esta codificada en C++ este plugin es absolutamente necesario para el desarrollo. Además el uso de las librerías de OpenCV y el compilador de desarrollo nativo en Android requieren configurar el proyecto con los menús anteriormente citados.

3.3. Android y SDK

Android es el sistema operativo objetivo de la aplicación. Este sistema es un SO libre para dispositivos móviles, como smartphones o tablets, basado en el kernel de Linux y desarrollado por Android y posteriormente comprado y soportado por Google.

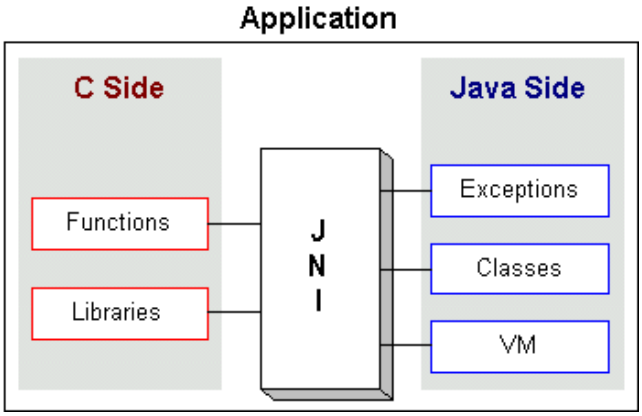
Para llevar a cabo el desarrollo de aplicaciones en Android necesitaremos el SDK (Software Development Kit) que nos provee la API y librerías necesarias para el desarrollo de nuestra aplicación. A fecha de este escrito existen 21 versiones de la API de Android.

En el caso de este trabajo el SDK es indispensable para la definición de la interfaz de usuario, la interacción con la cámara y el acceso al sistema de ficheros para las imágenes de calibración.

3.4. Java JNI y NDK de Android

En general el desarrollo de aplicaciones Android se lleva a cabo en java, sin embargo, cuando la eficiencia es una prioridad, como es el caso, existen herramientas para desarrollar en C y C++, es lo que se llama desarrollo nativo.

JNI (Java Native Interface) es un framework que nos permite que un programa escrito en Java interactúe con otro escrito en C, C++ u otros lenguajes.



Representación de la función de JNI

El NDK es un paquete de herramientas, similar al SDK, para Android que contiene entre otras cosas las librerías necesarias para desarrollar aplicaciones en Android con una parte nativa en C o C++ y un compilador que nos permite acceder a estas partes nativas desde el código Java al compilarlas como librerías.

Además del propio paquete NDK Android ofrece un plugin para Eclipse que nos facilita la compilación de nuestra parte nativa con el NDK.

Para este trabajo es necesario tener el paquete NDK para compilar las partes nativas del programa y también usare el plugin de eclipse, de forma que la compilación y ejecución del proyecto completo sea rápida y sencilla.

3.5. OpenCV

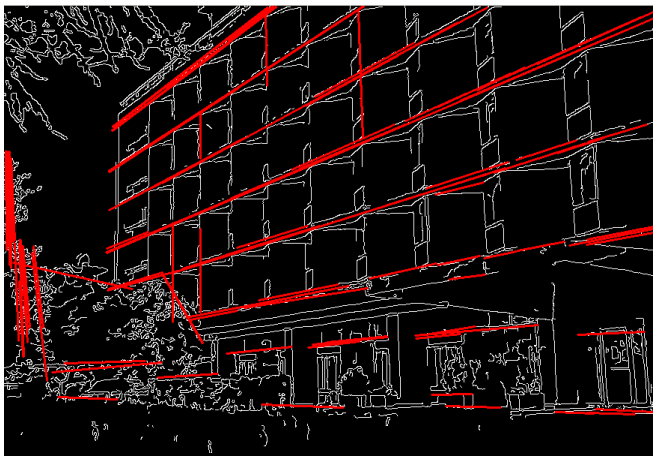
OpenCV es una biblioteca libre de visión artificial desarrollada originalmente por Intel. Tiene una amplia gama de funcionalidades como calibración de cámaras, visión estérea, reconocimiento facial, reconocimiento de patrones o filtros de imagen.

OpenCV es multiplataforma, tiene versiones para GNU/Linux MAC OS X, Windows y Android. También podemos encontrar su API para varios lenguajes de programación, concretamente, Python, C++ y C para ordenadores y Java en su versión Android.

Debido a su optimización, que hace de OpenCV una herramienta muy eficiente, y a su licencia BSD que permite, que sea usada libremente tanto para propósitos comerciales como de investigación, OpenCV goza de un buen reconocimiento y es usado en numerosos proyectos como el sistema de visión del vehículo no tripulado Stanley de la Universidad de Stanford.

Su papel en este trabajo es el de realizar la calibración de la cámara y procesar las imágenes para reconocer polígonos que puedan coincidir con las marcas que buscamos. Además, proporciona herramientas que nos permitirán calcular la posición relativa de dichas marcas respecto a la posición de la cámara.

En el siguiente capítulo se profundiza más en el tema de visión por computador y en el rol concreto que desempeña esta biblioteca en el trabajo. Incluyendo definiciones de funciones OpenCV y conceptos como la Matriz intrínseca o el vector de rotación.



Ejemplo de imagen procesada con OpenCV

4. CONCEPTOS Y TEORIA

4.1 Introducción a la visión por computador

La visión por computador es un amplio campo que consiste principalmente en métodos para analizar y procesar imágenes o elementos multidimensionales obtenidos en el mundo real en datos útiles. Las imágenes pueden ser obtenidas de muchas formas, como podría ser una sola imagen tomada con una cámara fotográfica o una secuencia de imágenes grabadas con una videocámara. Las aplicaciones de la visión por computador son muy extensas, por ejemplo reconocimiento facial, reconstrucción de imágenes, detección de movimientos o la propia realidad aumentada que trato en este trabajo.

Existen campos estrechamente ligados a la visión por computador como son la inteligencia artificial, la estadística y la geometría. Por ejemplo, mediante la visión por computador se pueden procesar imágenes obtenidas por una videocámara instalada en un robot, de forma que el procesado mediante técnicas de visión por computador facilite el aprendizaje de la inteligencia artificial de dicho robot. Las técnicas citadas pueden incluir elementos estadísticos como podría ocurrir en el caso de que el procesado de las imágenes incluyese clasificaciones de los objetos que en estas aparecen. Las técnicas de visión por computador nos ayudan a decidir si un elemento es suficientemente similar a otro previamente aprendido como para considerarlo una instancia de ese tipo de objeto.

4.1.1 Rol en este trabajo

El rol de la visión por computador en este trabajo es el de tratar las imágenes capturadas por la videocámara del dispositivo móvil, procesarlas para detectar si nuestra marca está presente en la imagen, calcular su posición en el espacio real y mostrarnos la información.

La calibración es un proceso que tiene como entrada una serie de fotografías de un patrón conocido tomadas por la cámara que deseamos calibrar y como salida una serie de parámetros que definen las distorsiones que aplica la cámara sobre la imagen, por ejemplo al tener una lente curvada o un gran angular. Estos parámetros son únicos para esa cámara y esa distancia focal concreta y pueden ser reusados en tanto en cuanto no se modifique la distancia focal.

Estos parámetros son conocidos como parámetros intrínsecos. Los parámetros intrínsecos se representan en una matriz de 3X3 conocida como la matriz intrínseca y en un vector de hasta 8X1 (también 4X1 y 5X1) conocido como vector de coeficientes de distorsión.

Estos parámetros se usan para localizar o proyectar puntos en tres dimensiones en el plano usando la perspectiva. La ecuación general es la siguiente:

$$sm' = A[R|t]M'$$

O bien:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r11 & r12 & r13 & t1 \\ r21 & r22 & r23 & t2 \\ r31 & r32 & r33 & t3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Donde:

- {X,Y,Z} son las coordinadas de un punto 3D en el espacio de coordenadas real
- {u,v} son las coordinadas de proyección del punto en pixeles
- A es la matriz intrínseca de la cámara
- {cx,cy} es el punto principal que es normalmente el punto central de la imagen
- {fx,fy} son las distancias focales expresadas en pixeles.

Por lo tanto si una imagen de la cámara calibrada es reescalada por una constante, los parámetros de la matriz intrínseca deben ser reescalados de la misma manera, ya sea dividiendo o multiplicando.

La unión de las matrices de rotación y traslación [R|t] es conocida como la matriz de parámetros extrínsecos. Dicha matriz es única para cada imagen diferente y describe el movimiento de una cámara alrededor de un objeto o viceversa el movimiento de un objeto delante de una cámara. La matriz R describiría la rotación del objeto, mientras que la matriz t describiría la traslación. Para z≠0 la anterior transformación sería equivalente a:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

$$x' = \frac{x}{z}$$

$$y' = \frac{y}{z}$$

$$u = fx * x' + cx$$

$$v = fy * y' + cy$$

Sin embargo, también debemos tener en cuenta los otros parámetros intrínsecos, los correspondientes a la distorsión. Generalmente las lentes tienen algo de distorsión, mayormente distorsión radial y una ligera distorsión tangencial. Por lo tanto añadiendo estos parámetros al modelo tendríamos el siguiente resultado.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

$$x' = \frac{x}{z}$$

$$y' = \frac{y}{z}$$

$$x'' = \frac{x'(1 + k1r^2 + k2r^4 + k3r^6)}{(1 + k4r^2 + k5r^4 + k6r^6)} + 2p1x'y' + p2(r^2 + 2x'^2)$$

$$y'' = \frac{y'(1 + k1r^2 + k2r^4 + k3r^6)}{(1 + k4r^2 + k5r^4 + k6r^6)} + p1(r^2 + 2y'^2) + 2p2x'y'$$

$$u = fx * x'' + cx$$

$$v = fy * y'' + cy$$

k1, k2, k3, k4, k5 y k6 son coeficientes de distorsión radial y p1 y p2 son coeficientes de distorsión tangencial. En OpenCv hay un máximo de ocho coeficientes de distorsión y un mínimo de cuatro, contando siempre como mínimo con 2 coeficientes de distorsión radial y 2 de distorsión tangencial, es decir siempre que aumentamos el número de coeficientes obtenemos coeficientes de distorsión radial.

Como se puede observar la corrección de las distorsiones añade un paso intermedio a la transformación al espacio plano o bidimensional.

Al contrario que la matriz intrínseca los coeficientes de distorsión son independientes del tamaño de la imagen con lo que unos parámetros obtenidos para una imagen de resolución 320 x 240 son perfectamente válidos para usar en una imagen de 640 x 480 sin necesidad de modificarlos.

En resumen la matriz intrínseca nos permite transformar coordinadas reales al plano bidimensional de la imagen y los coeficientes de distorsión se encargan de reajustar esas coordenadas en el plano conforme a los las distorsiones provocadas por la lente.

4.1.2 OpenCv

Ahora voy a situar los conceptos aprendidos en el apartado anterior en su contexto, es decir, como encajan en OpenCv y en nuestro sistema. Para eso voy a exponer las definiciones de las 2 funciones clave de openCv que nos permiten conocer los

parámetros intrínsecos y posteriormente usar esos parámetros para localizar elementos en el espacio.

La primera función de OpenCv en la que me voy a centrar es calibrateCamera, cuya definición es la siguiente:

calibrateCamera

Encuentra los parámetros intrínsecos y extrínsecos de varias vistas de un patrón de calibración.

C++: double `calibrateCamera`(InputArrayOfArrays **objectPoints**, InputArrayOfArrays **imagePoints**, Size **imageSize**, InputOutputArray **cameraMatrix**, InputOutputArray **distCoeffs**, OutputArrayOfArrays **rvecs**, OutputArrayOfArrays **tvecs**, int **flags**=0, TermCriteria **criteria**=TermCriteria(TermCriteria::COUNT+TermCriteria::EPS, 30, DBL_EPSILON))

Donde:

- objectPoints: es un array de arrays de puntos que se corresponden con las coordenadas en el mundo real del patrón que usamos para calibrar la cámara. Como el patrón es siempre el mismo todos los arrays del array contienen los mismos valores. Es decir en nuestra anterior ecuación cada punto seria la variable $\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$.
- imagePoints: es un array de arrays de puntos que se corresponden con los puntos de objectPoints transformados en la imagen, es decir son bidimensionales y se corresponderían con la variable $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$.
- imageSize: en el caso del patrón de tablero de ajedrez que es el que uso en el trabajo para calibrar la cámara, este parámetro define el número de filas y columnas que tiene el patrón.
- cameraMatrix: es la matriz intrínseca calculada por la función y es el primero de los parámetros de salida. En la ecuación anterior se corresponde con la variable [R|t].
- disCoeffs: es el vector de coeficientes de distorsión, cuyo tamaño puede variar de 4 a 8 elementos (4,5 u 8 elementos). En la ecuación anterior se corresponde con los valores {k1, k2, p1, p2, [k3, [k4, k5, k6]]} según su longitud.
- rvecs: es el vector de rotación que representa en cada imagen la rotación del patrón relativa a la cámara. Es diferente para cada imagen usada en la calibración.

- tvecs: es el vector de traslación que representa en cada imagen la rotación del patrón relativa a la cámara. Es diferente para cada imagen usada en la calibración.
- flags: parámetro opcional que define diferentes opciones para la calibración como establecer a cero los coeficientes de distorsión tangencial. Este parámetro no se usa en el trabajo.
- criteria: Parámetro opcional que define el criterio de terminación del algoritmo de optimización. Este parámetro no se usa en el trabajo.
- term_crit: Igual que el parámetro anterior.

Como se puede observar esta función será la que nos provea los parámetros intrínsecos necesarios para calibrar nuestra cámara.

La siguiente función se encarga de localizar el elemento que buscamos en el espacio. Su nombre es solvePnP y su definición es la siguiente:

solvePnP

Encuentra la posición de un objeto de su correspondencia 3D-2D.

C++: bool `solvePnP`(InputArray **objectPoints**, InputArray **imagePoints**, InputArray **cameraMatrix**, InputArray **distCoeffs**, OutputArray **rvec**, OutputArray **tvec**, bool **useExtrinsicGuess**=false, int **flags**=ITERATIVE)

Donde:

- objectPoints: Es un array de puntos que representa los vértices del elemento en el espacio real (en 2 dimensiones)
- imagePoints: Es un array de puntos que representan los vértices del elemento en la imagen (en 2 dimensiones).
- cameraMatrix: La matriz de parámetros intrínsecos de la cámara [R|t] que permite transformar puntos del sistema de referencia del espacio real al sistema de referencia del plano o imagen y viceversa.
- disCoeffs: El vector de coeficientes de distorsión usados para corregir las distorsiones radiales y tangenciales.
- rvec: Vector que representa la rotación relativa del elemento respecto a la cámara.
- tvec: Vector que representa la traslación relativa del elemento respecto a la cámara.
- useExtrinsicGuess: Si es true, esta variable usa los valores de rvec y tvec que le proporcionemos como aproximaciones iniciales a estos vectores y posteriormente los optimiza.

- flags: Define el método para resolver el problema PnP. Hay 3 opciones, en el caso de este trabajo usamos el método por defecto que es el método iterativo basado en la optimización de Levenberg-Marquardt.

Con los vectores rvec y tvec obtenemos la posición en el espacio real, que es el resultado final que deseamos y que nos permitirá, junto a los parámetros intrínsecos, proyectar objetos en la imagen con la perspectiva adecuada, y por lo tanto, aumentar la realidad.

4.2 Android

En este apartado vamos a ver conceptos relacionados con Android y con un proyecto como el que se desarrolla en este trabajo.

El primer concepto que vamos a revisar es el de Activity. Una Activity es una parte de una aplicación que contiene la lógica de una de las "pantallas" de la aplicación. Cada Activity se corresponde con una clase en Java que extiende a la clase Activity. La aplicación de este trabajo tendrá dos Activities, una para la calibración y otra para la localización.

Por otra parte tenemos el Android Manifest o Manifest a secas. Este archivo es un archivo de tipo XML que contiene información sobre la aplicación. El sistema usa este archivo para saber que existe la aplicación y comprender sus componentes. Además, tiene otras funciones como:

- Identificar los permisos de usuario que requiere la aplicación.
- Declarar el nivel de API mínimo que requiere la aplicación.
- Declara requisitos de software y hardware, como la cámara que en este caso necesitamos.
- Librerías a las que necesita conectarse la aplicación a parte de las APIs de Android en sí mismas. En este trabajo, por ejemplo, necesitamos las librerías de OpenCv.

Como hemos dicho este documento informa al sistema de sus componentes, es decir de sus Activities y sus propiedades y requisitos.

El siguiente concepto es el de layout. Un layout se puede definir como la parte grafica de una Activity. Generalmente cada Activity va asociada a un layout y este proporciona la interfaz de usuario mediante la que la Activity se comunica. En este trabajo tendremos dos layout, una para cada Activity.

5. DISEÑO DE LA APLICACION

5.1 Descripción del problema

El objetivo del trabajo es el diseño e implementación de una aplicación para dispositivos móviles que sirva como base a futuras aplicaciones de realidad aumentada.

Este trabajo deberá poder ser usado como práctica docente, de forma que la memoria debe incluir un tutorial para facilitar la implementación de un sistema similar y un manual de la propia aplicación.

La aplicación se desarrolla en Android, dado que es para dispositivos móviles. Debido a esto el uso de java es mandatorio. Además, la aplicación hará uso también de desarrollo nativo en C++ para aprovechar su eficiencia.

La aplicación consta de dos funciones principales, la calibración de la cámara y la localización de la posición donde deseamos proyectar las imágenes que nos permitirán aumentar la realidad.

5.2 Diseño

5.2.1 Calibración

La calibración es el primer módulo o la primera parte de la aplicación que debía desarrollar.
Esta parte debería tener como input una serie de imágenes que permitieran calibrar la cámara, realizar internamente la calibración de la cámara, almacenar los parámetros obtenidos y ceder el control de la aplicación a otra actividad encargada del resto de funcionalidades de la aplicación, es decir, de reconocer la marca, calcular sus parámetros de posición y mostrárselos al usuario.

Lo primero que debía decidir era como se obtenían las imágenes que nos permitirían calibrar la cámara. Consideré el estudio de tres opciones:

1. Usar las imágenes almacenadas en la carpeta “Res” del proyecto de Android. Esta carpeta suele contener los recursos gráficos de un proyecto.
2. Obtener las imágenes en tiempo real, tomando fotos mediante la cámara del móvil y procesándolas en ese momento. Todo como parte de la aplicación.
3. Leer las imágenes de una carpeta en la memoria del teléfono.

Después de estudiar las opciones llegue a las siguientes conclusiones:

La primera opción la descarté dado que para acceder a la carpeta Res del proyecto necesitaríamos tener acceso al código fuente y por tanto al proyecto en sí mismo. Además de ser menos accesible que cualquiera de las otras dos opciones.

La segunda opción es viable y bastante buena, sin embargo requiere un esfuerzo extra en el sentido de que la actividad de calibración debe constar de una capturadora de imágenes, es decir, una cámara para tomar las fotos pertinentes.

La tercera opción tiene el inconveniente de que, en caso de que la carpeta no sea la carpeta por defecto donde se almacenan las fotografías, una vez hechas deben moverse a la carpeta asignada a las imágenes usadas en la calibración. Ya sea mediante una aplicación móvil que explore las carpetas o conectando el dispositivo móvil a un ordenador y usando el propio explorador.

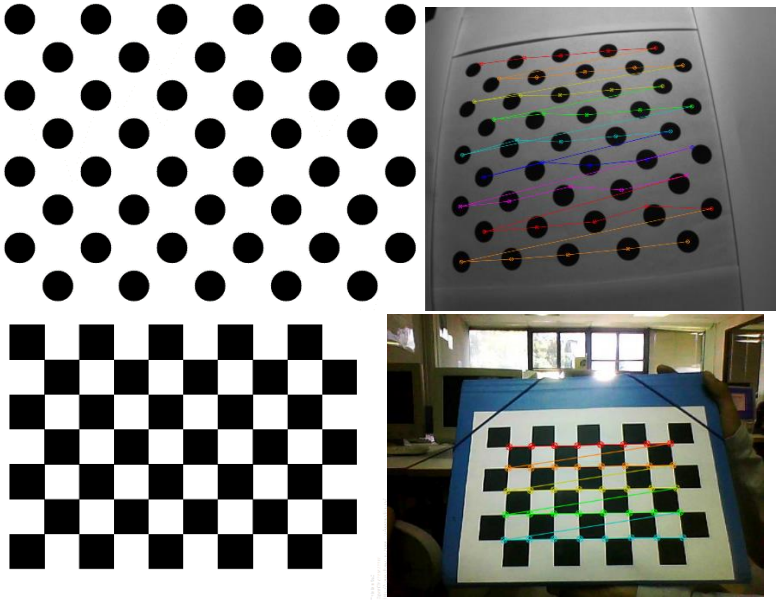
En el caso de usar la carpeta por defecto donde se almacenan fotografías debemos tener en cuenta que es probable que haya numerosas fotografías no relacionadas con la calibración y no queremos tener que procesarlas todas puesto que sería computacionalmente ineficiente e innecesario. De este modo lo ideal sería asignar una carpeta propia a las imágenes destinadas a la calibración y por lo tanto sería necesario moverlas.

En base a estas conclusiones deduje que la mejor opción es una combinación de la segunda y la primera. En este supuesto la cámara captura las imágenes y las almacena en el almacenamiento interno del sistema en una carpeta exclusiva para las imágenes de calibración.

Sin embargo para un funcionamiento simple del sistema nos basta con la tercera opción, almacenar las imágenes en una carpeta exclusiva de las imágenes de calibración. La implementación de la captura de imágenes para calibración dentro de la propia aplicación puede considerarse una ampliación para el futuro.

A la hora de decidir el patrón que usaría para la calibración me encontré las siguientes posibilidades:

1. Usar un patrón de círculos, cuya detección viene soportada ya por las librerías de OpenCv.
2. Usar un patrón de tablero de ajedrez que, como el patrón de círculos, tiene una función específica definida en las librerías de OpenCv que resuelve el problema.
3. Diseñar un patrón de calibrado e implementar una función que procesase las imágenes para encontrar los puntos relevantes.



Imágenes ideales de los patrones (izq) e imágenes post-procesadas (dcha.)

La tercera posibilidad es con diferencia la más complicada y la menos eficiente puesto que debería decidir cómo sería el patrón, cuáles serían los puntos de calibración e implementar la detección de dichos puntos.

De las otras dos opciones podría haber elegido cualquiera indistintamente. Sin embargo, dado que ya había trabajado con el patrón de ajedrez (ver archivo CalPattern.png) y estaba familiarizado con él, decidí volver a usarlo para este trabajo.

La tercera parte de la calibración que requería decisiones de diseño es la forma en que se almacenan los parámetros para ser cargados, tanto por la actividad encargada de la localización como en sesiones posteriores.

En este caso exploré dos opciones:

1. Almacenar la matriz de la cámara y los coeficientes de distorsión en un archivo de texto corriente con un formato específicamente diseñado para este propósito.
2. Almacenar la matriz de la cámara y los coeficientes de distorsión en un archivo XML usando la clase SharedPreferences que implementa Android usada comúnmente para guardar las opciones de ajustes de las aplicaciones.

En esta ocasión ambas opciones son sencillas de implementar y perfectamente válidas. Además al ser transparentes al usuario hay que tener muchas menos consideraciones. Por este motivo la opción que elegí se corresponde con la más cómoda de implementar que, en este caso, es la segunda.

La clase SharedPreferences provee una funcionalidad que se adapta perfectamente a este caso, permite guardar los parámetros relevantes con una etiqueta con la que luego podemos cargarlos de nuevo en otra actividad u otra sesión de la aplicación. Con esto evitamos complicar el código leyendo un archivo de texto y diseñando un formato específico para este, como sería el caso de la primera opción.

5.2.2 Localización

El módulo de localización es el segundo y último modulo del trabajo y se corresponde con una segunda actividad en la aplicación Android.

Este módulo consta de una cámara de video que recoge los fotogramas que serán posteriormente procesados para encontrar y localizar el patrón que estamos usando. Además nos proveerá por pantalla de los resultados del proceso.

Para procesar los fotogramas necesitamos los parámetros intrínsecos previamente calculados. Como expliqué en el apartado anterior estos parámetros están almacenados en un fichero XML de preferencias y lo primero que debemos hacer es cargarlos y activar la cámara.

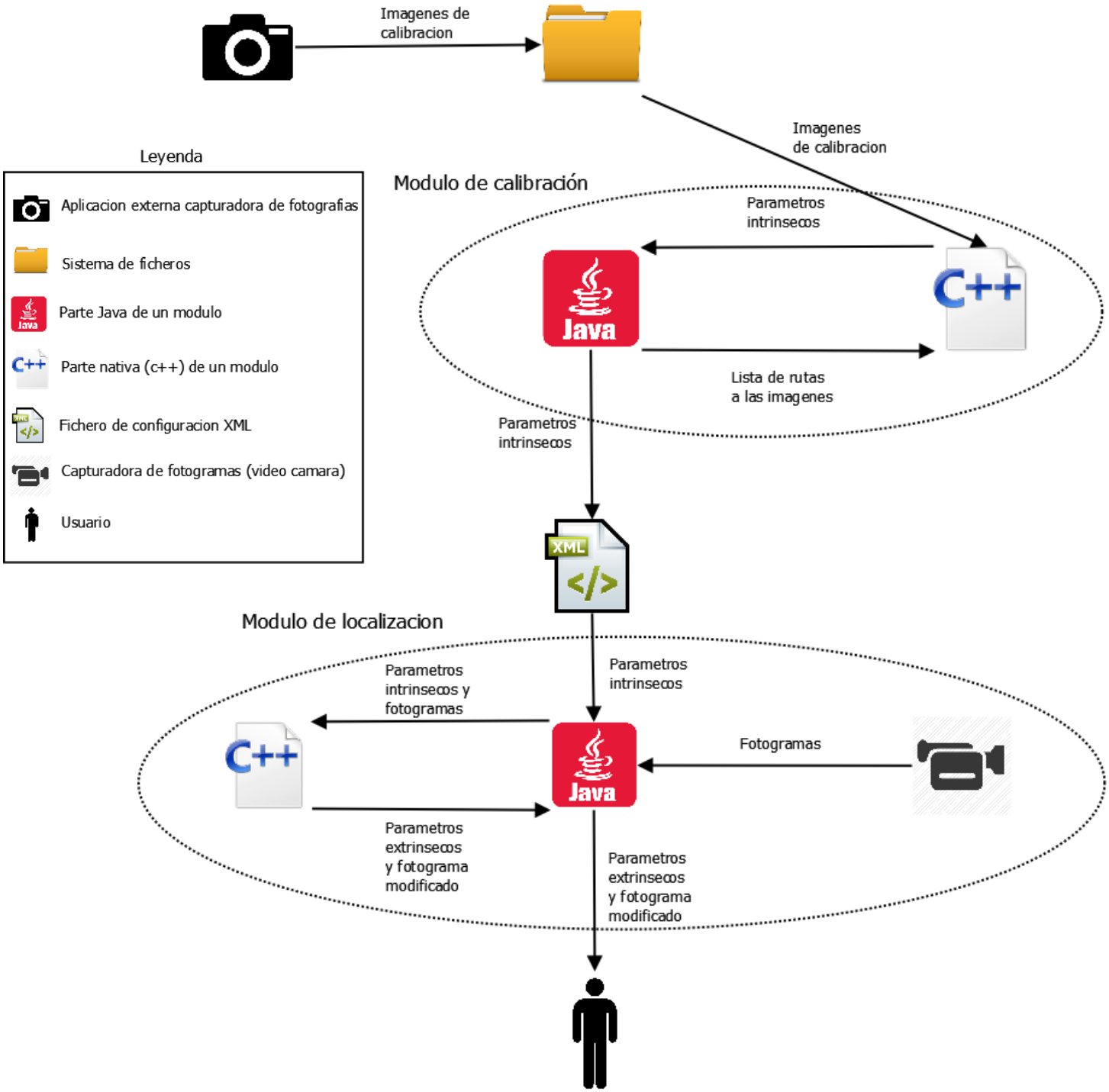
El patrón de localización (ver archivo LocPattern.pdf) consiste en 2 cuadrados concéntricos cuyos lados miden 150mm y 75mm respectivamente. El espacio entre ambos cuadrados es negro y tanto el exterior como el interior del cuadrado más pequeño es blanco.

La elección de este patrón se debe a que ya había trabajado con él y el algoritmo de reconocimiento es relativamente sencillo.

Luego procesaremos los datos usando las funciones definidas en la librería nativa (c++). Finalmente mostraremos los resultados en tiempo real en la pantalla del dispositivo. Por la pantalla se mostrara, tanto el fotograma con el contorno del patrón dibujado, en caso de ser detectado, como los parámetros extrínsecos calculados en base a la posición del patrón de localización.

5.2.3 Diagrama de flujo de información

Para facilitar la comprensión del funcionamiento del sistema he desarrollado un diagrama de flujo que expone el movimiento de la información a través de las partes involucradas en el sistema.



5.2.4 Interfaz de usuario

Para la interfaz de usuario no requiere nada complejo. Como he mencionado anteriormente la aplicación consta de 2 módulos y a cada una le corresponde una actividad por lo que debo diseñar dos pantallas, o layouts, una para cada actividad.

La primera pantalla contendrá únicamente dos botones. El primero de ellos al pulsarlo debería calibrar la cámara con las imágenes almacenadas y posteriormente permitiría pasar a la siguiente actividad. El segundo servirá para borrar la configuración de calibración anterior y poder realizar una nueva calibración, por ejemplo si hemos añadido o modificado las fotografías que usamos en la calibración.

En la segunda actividad, la que corresponde al módulo de localización, tendremos una interfaz constituida únicamente por una superficie que muestre los fotogramas de la cámara después del procesado.

6. DESARROLLO

Este capítulo contiene apartados en los que se incluyen fragmentos de código. Algunos de estos fragmentos pueden ser difíciles de ubicar por lo que es recomendable tener una copia del código para revisar y poder colocar cada fragmento en su contexto.

6.1 Instalación del entorno

Para instalar y configurar el entorno necesitaremos descargar primero cierto software. Podemos encontrar distribuciones para diferentes sistemas y arquitecturas en los siguientes links:

(Todos los links son enlaces a las últimas versiones en la fecha en que se escribió esta memoria, si alguno de ellos no se encuentra disponible por favor dirígete a la página oficial.)

Java JDK
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Eclipse con el plugin CDT
https://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/lunasr1

Android SDK(SDK tools only)
http://developer.android.com/sdk/index.html#Other

Android NDK
https://developer.android.com/tools/sdk/ndk/index.html

Android OpenCv SDK
http://sourceforge.net/projects/opencvlibrary/files/opencv-android/2.4.10/OpenCV-2.4.10-android-sdk.zip/download

A continuación se describen los pasos necesarios para instalar un entorno que nos permita programar, compilar y ejecutar un proyecto de aplicación de Android con parte nativa.

Para usar eclipse y poder programar en Android con java lo primero que debemos instalar es el jdk de Java. Podemos encontrar la distribución de nuestro sistema operativo en el siguiente enlace:

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Si usamos Ubuntu también tenemos la posibilidad de instalarlo desde la línea de comandos con el comando:

```
sudo apt-get install default-jdk
```

Una vez hemos hecho esto descomprimimos eclipse y lo abrimos. El eclipse que nos hemos descargado ya tiene instalado el plugin que incluye las configuraciones para C++ y el editor de C++. Por lo tanto ahora debemos instalar el plugin adt siguiendo estos pasos:

1. Abre eclipse y clica **en Help->Install New Software**
2. Clic en **Add** en la esquina superior derecha.
3. En la siguiente ventana que aparece pon "ADT plugin" en la casilla Name y la siguiente URL en Location:

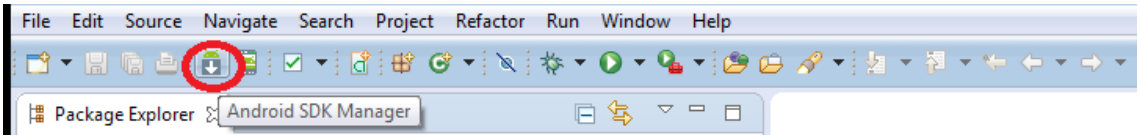
   ```
   https://dl-ssl.google.com/android/eclipse/
   ```

4. Haz clic en **OK**
5. En el cuadro de Available Software selecciona la casilla al lado de Developer Tools y haz clic en **Next**.
6. En la siguiente ventana veras una lista de las herramientas que se van a descargar. Haz clic en **Next**.
7. Lee y acepta los acuerdos de licencia y haz clic en **Finish**.
8. Cuando se termine de instalar, reinicia eclipse.

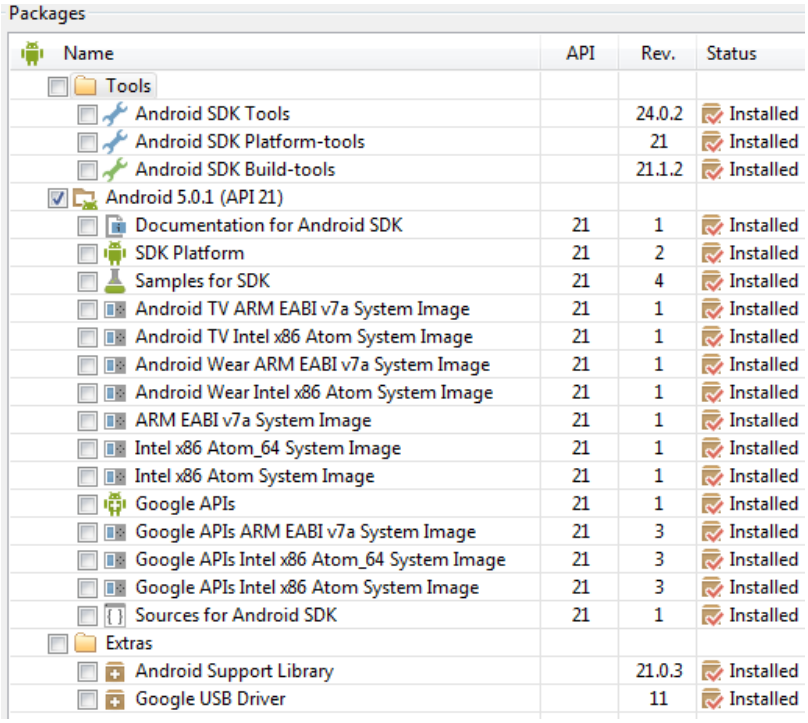
Cuando eclipse se reinicie te pedirá que especifiques la ubicación del SDK de Android. Descomprime el Android sdk y copia la ruta para indicársela a eclipse.

Te aparecerá un mensaje pidiendo abrir el SDK manager. Ábrelo.

Si este mensaje no aparece cambia la perspectiva de eclipse de C/C++ a java en la esquina superior derecha y haz clic en el botón de la imagen que se encuentra debajo. Recuerda que el botón solo aparece en la perspectiva de java:



Una vez en el SDK Manager instala los siguientes paquetes:



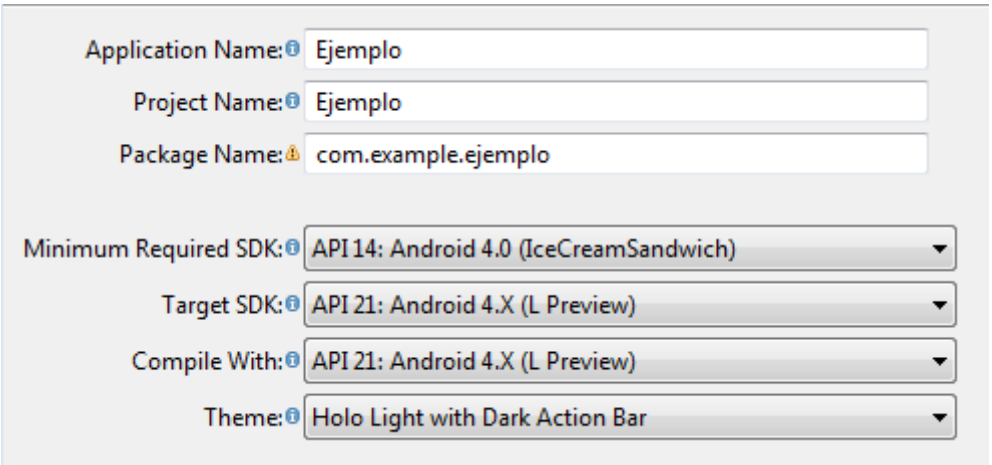
Finalmente configuramos la ruta del ndk. Para ello hacemos clic en **Window->Preferences->Android->NDK** y en la casilla NDK Location introducimos la ruta del Android NDK que hemos descargado previamente.

6.2 Crear y configurar un proyecto

Ahora vamos a crear un proyecto de aplicación Android y a configurarlo para que compile incluyendo la parte nativa o c++ que hayamos codificado.

Hacemos clic en **File->New->Android Aplication Project**

Damos nombre a la aplicación y elegimos las versiones de android para las que vamos a desarrollarla, tanto la mínima requerida como la versión objetivo y hacemos clic en **Next**.



En las siguientes ventanas hacemos clic en **Next** hasta llegar a la última donde hacemos clic en **Finish**.

Una vez creado el proyecto hacemos clic derecho en este en el Package Explorer y vamos a **Andoid tools->Add native support…**

Esto nos abrirá una ventana donde podemos indicar el nombre de la librería que vamos a definir en C/C++. Escribimos el nombre que deseemos por ejemplo "Ejemplo" y hacemos clic en **Finish**.

Ahora nuestro proyecto contiene una carpeta "jni" en la que tenemos 2 archivos, Android.mk y Ejemplo.cpp. El archivo Ejemplo.cpp es donde se ubicara el código C++ que queramos compilar como una librería para usar posteriormente en Java.

Para añadir las librerías de OpenCv al proyecto vamos a **File->Import** y se nos abrirá una ventana donde elegimos **Android->Existing Android Code into Workspace** y hacemos clic en **Next**.

En la siguiente pantalla hacemos clic en **Browse** y vamos a "ruta_openCV_Android"/sdk/ y elegimos la carpeta "Java". Abajo nos saldrá Un proyecto, si la casilla que aparece no está marcada la marcamos y hacemos clic en **Finish**.

Ahora hacemos clic derecho en nuestro proyecto y en **Properties**. Vamos a la pestaña Android y en la parte de la derecha hacemos clic en **Add…** Nos aparecerá una ventana donde está el proyecto OpenCv, marcamos la casilla y hacemos clic en **OK**.

Con esto tenemos configurada la parte java. Ahora para configurar la parte C++ vamos a **Project->Properties** y a la pestaña C/C++ Build. Desmarcamos la casilla Use Default build command y en Build command escribimos "ndk-build", si esto no funciona escribe la ruta completa al ndk-build (En Windows puede que necesites poner la extensión ".bat") y finalmente hacemos clic en **OK**.

Para agregar las librerías de OpenCv a la parte de C++ editamos el archivo Android.mk y después de include $(CLEAR VARS) agregamos la línea:

include {ruta_OpenCv_Android_sdk}/sdk/native/jni/OpenCV.mk

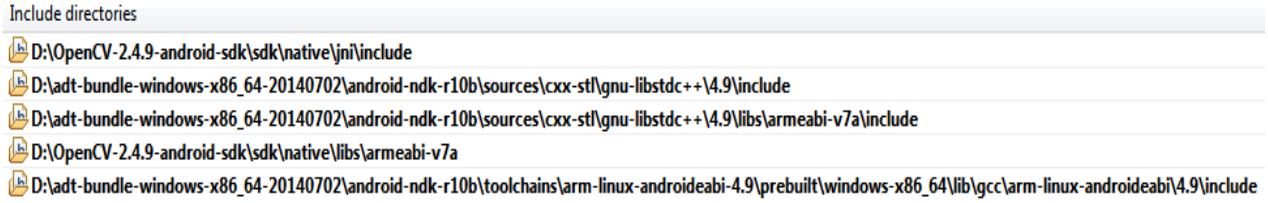
Obteniendo un resultado como este:

```
1 LOCAL_PATH := $(call my-dir)
2
3 include $(CLEAR_VARS)
4
5 include D:/OpenCV-2.4.9-android-sdk/sdk/native/jni/OpenCV.mk
6
7 LOCAL_MODULE    := Prueba
8 LOCAL_SRC_FILES := Prueba.cpp
9 LOCAL_LDLIBS +=  -llog -ldl
10
11 include $(BUILD_SHARED_LIBRARY)
12
```

Más adelante veremos el contenido de este archivo con detenimiento.

NOTA: Estos últimos pasos difieren mucho de una versión a otra de eclipse y puede que no sean los adecuados para arreglar los errores en el indexado del editor o en la compilación. Todos estos pasos funcionaron en Eclipse Juno 4.2.2

Ahora para agregar las librerías que nos deberían permitir arreglar los errores que nos aparecen en el editor vamos a C/C++ General-> Paths and symbols y en la pestaña "includes" agregamos estas 5 rutas:



Hacemos clic en Apply y nos pedirá recompilar, aceptamos y hacemos clic en OK.

De esta forma nuestro proyecto debería ser capaz de compilar e indexar correctamente.

6.3 Andorid.mk, Application.mk y Manifest

Como se explica en el apartado de conceptos, a la hora de desarrollar una aplicación no solo debemos programar los archivos fuente en java y c++. También debemos definir ciertos archivos para que nuestra aplicación funcione. En concreto debemos definir el

AndroidManifest.xml en todas nuestras aplicaciones y aquellas que tengan una parte nativa, como la que nos ocupa, debemos definir el Android.mk y en ocasiones el Application.mk.

En este apartado vamos a explorar los contenidos de estos archivos en este trabajo.

6.3.1 Android.mk

Este archivo es definido para describir nuestros recursos al sistema de compilación. En este archivo declaramos las librerías estáticas o compartidas que usaremos en la aplicación. Las librarías compartidas serán copiadas/instaladas en el paquete de la aplicación. En este trabajo compilaremos nuestro modulo en C++ como una librería compartida puesto que queremos que se instale en nuestra aplicación.

Puedes declarar uno o varios módulos en el Android.mk, en este caso solo tenemos un módulo, el modulo NativeLib.

El contenido de este archivo en la aplicación es:

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

include D:/OpenCV-2.4.9-android-sdk/sdk/native/jni/OpenCV.mk

LOCAL_MODULE:= NativeLib
LOCAL_SRC_FILES := NativeLib.cpp
LOCAL_LDLIBS +=-llog -ldl

include $(BUILD_SHARED_LIBRARY)
```

Veamos el propósito de las líneas una a una:

```
LOCAL_PATH := $(call my-dir)
```

La primera línea asigna un valor a la variable LOCAL_PATH que debe encontrarse en el principio de todos los archivos Android.mk. En este caso le asignamos el valor de la ruta en la que se encuentra el propio archivo llamando a la función del sistema de compilación my-dir.

```
include $(CLEAR_VARS)
```

Esta línea incluye una variable que nos provee el sistema de compilación y que limpia los valores de todas las variables LOCAL_XXX menos la variable LOCAL_PATH. Es necesario hacer esto puesto que todos los ficheros de control de la compilación son

parseados en una sola ejecución de make y las variables son globales con lo que no queremos tener conflictos.

```
include D:/OpenCV-2.4.9-android-sdk/sdk/native/jni/OpenCV.mk
```

Incluimos la ruta del mk de OpenCv para poder compilar el modulo que hace uso de OpenCv.

```
LOCAL_MODULE:= NativeLib
```

Indicamos el nombre del módulo que vamos a compilar. Este será el nombre con el que carguemos la librería posteriormente en Java.

```
LOCAL_SRC_FILES := NativeLib.cpp
```

A continuación definimos el archivo fuente que queremos compilar para obtener la librería en LOCAL_SRC_FILES.

```
LOCAL_LDLIBS +=-llog -ldl
```

Añadimos en LOCAL_LDLIBS las librerías que nos permiten usar el logcat de Android desde C/C++

```
include $(BUILD_SHARED_LIBRARY)
```

BUILD_SHARED_LIBRARY es, como CLEAR_VARS, una variable que provee el sistema de compilación y que apunta a un script que esta al cargo de recopilar la información que has definido en las variables LOCAL_XXX desde el ultimo $(CLEAR_VARS). Su equivalente para librerías estáticas es BUILD_STATIC_LIBRARY

6.3.2 Application.mk

Este archivo complementa al Android.mk y su propósito es describir que módulos nativos necesita la aplicación. El contenido es el siguiente:

```
APP_STL := gnustl_static
APP_CPPFLAGS := -frtti -fexceptions
APP_ABI := armeabi-v7a
APP_PLATFORM := android-8
```

Veámoslo línea por línea:

```
APP_STL := gnustl_static
```

Esta línea define la implementación de la runtime library de C++ que deseamos usar en nuestra aplicación. En este caso usamos la librería gnustl estática.

```
APP_CPPFLAGS := -frtti –fexceptions
```

Esta variable contiene flags para la compilación de las librerías en c++. -frtti añade activa el RTTI (RunTime Type Information) que nos permite exponer información sobre objetos en tiempo de ejecución. –fexceptions permite el manejo de excepciones.

```
APP_ABI := armeabi-v7a
```

APP_ABI define el ABI (Aplication binary interface) para el que el compilador genera el código máquina. Esto es el set de instrucciones que usa nuestra arquitectura. En este caso nos interesan los dispositivos ARMv7 por lo que elegimos su ABI correspondiente.

```
APP_PLATFORM := android-8
```

Aquí definimos la plataforma mínima de Android que soportará nuestro código nativo y debe coincidir con la definida en el Manifest.

6.3.3 Android Manifest

Este archivo es más extenso por lo que vamos a revisarlo por partes.

Comenzamos por la cabecera:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="upm.tfg"
android:versionCode="1"
android:versionName="1.0" >
```

En este fragmento definimos las propiedades del documento xml, el name space, el nombre del paquete que contiene el código fuente, la versión del código y el nombre que queremos dar a esa versión.

```
<uses-sdk
android:minSdkVersion="8"
android:targetSdkVersion="21" />
```

A continuación definimos la versión mínima de Android que soportaría la aplicación y la versión para la que queremos desarrollarla. En este caso la versión mínima es la API 8 y la versión objetivo la 21, la última.

```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<uses-permission android:name="android.permission.CAMERA" />
```

Este fragmento define los permisos. Como vamos a tener que escribir en el almacenamiento para crear la carpeta de imágenes de calibración y guardar los parámetros intrínsecos necesitamos el permiso para escribir en el almacenamiento.

Además usaremos la cámara en la localización por lo que también necesitamos permiso para acceder a ella.

```
<supports-screens
android:anyDensity="true"
android:largeScreens="true"
android:normalScreens="true"
android:resizeable="true"
android:smallScreens="true" />
```

Confirmamos que la aplicación soporta todo tipo de pantallas, grandes, pequeñas, medianas, etc.

```
<uses-feature
android:name="android.hardware.camera"
android:required="false" />
<uses-feature
android:name="android.hardware.camera.autofocus"
android:required="false" />
<uses-feature
android:name="android.hardware.camera.front"
android:required="false" />
<uses-feature
android:name="android.hardware.camera.front.autofocus"
android:required="false" />
```

Declaramos las características de la cámara que usamos.

Entramos en el bloque de declaración de las actividades de la aplicación. Primero definimos algunas propiedades de esta.

```
<application
 android:allowBackup="true"
 android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"android:theme="@android:style/Th
      eme.Black.NoTitleBar.Fullscreen" >
```

android:allowBackup permite a la aplicación participar en los backups, por ejemplo si realizamos un backup del sistema.

android:icon define el icono que tendrá la aplicación. En este caso el icono esta en los recursos de la aplicación.

android:label establece el nombre de la aplicación que está declarado en los recursos de la aplicación nuevamente.

android:theme sirve para establecer un Tema por defecto en la interfaz. Además, en este caso eliminamos la barra de título de la aplicación para que se vea a pantalla completa.

Ahora vamos a ver las declaraciones de las Activities. Como sabemos hay dos Activities, la primera es la de Calibración:

```
<activity
android:name="upm.tfg.Calibracion"
android:configChanges="orientation|keyboardHidden"
android:label="@string/app_name"
android:screenOrientation="portrait" >
<intent-filter>
 <action android:name="android.intent.action.MAIN" />

 <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
```

Con android:name definimos el lugar donde se encuentra la Activity dentro del proyecto.

android:configChanges junto a android:screenOrientation obliga a la Activity a mostraste siempre en posición vertical o “portrait”.

android:label define el título de la Activity, es decir, es lo que se mostraría en la barra de título de la interfaz en caso de que se mostrase.

El fragmento que envuelve el <intent-filter> define las propiedades de los intents que realiza la Activity.

<action android:name/> define el nombre de la acción que realiza el intent. El nombre "android.intent.action.MAIN" es una acción estándar definida en la clase Intent.

<category android:name/> añade el nombre de la categoría y es un campo opcional. Igual que action, en este caso es un valor estándar.

El último fragmento del Manifes es la segunda Activity, Localización.

```
<activity
android:name="upm.tfg.Localizacion"
android:configChanges="keyboardHidden|orientation"
android:label="@string/app_name"
android:screenOrientation="landscape" >
</activity>
</application>
</manifest>
```

Los atributos de esta Activity son similares a los de la otra, sin embargo, en este caso forzamos la Activity a mostrarse siempre en la vista horizontal o "Landscape".

6.4 Interfaz de usuario "Layouts"

En este apartado vamos a ver los archivos XML que definen la interfaz de usuario de la aplicación. Estos archivos se encuentran en {proyecto}/res/layout y pueden apoyarse en otros recursos de /res.

Para empezar veamos el Layout de la Activity de calibración que he llamado calibración.xml.

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context=".MainActivity"

android:background="@drawable/background"
>

<LinearLayout
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:background="@drawable/linearlayout_bg"
android:orientation="vertical"
android:padding="10dp" >

<Button
android:id="@+id/btnCalibrar"
style="@style/DefaultButtonText"
```

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_margin="4dp"
android:background="@drawable/button_default_bg"
android:padding="10dp"
android:text="@string/boton" />

<Button
android:id="@+id/btnBorrar"
style="@style/DefaultButtonText"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_margin="4dp"
android:background="@drawable/button_default_bg"
android:padding="10dp"
android:text="@string/reset" />
</LinearLayout>

</RelativeLayout>
```

Para poder comprenderlo mejor vamos a verlo por bloques. Comenzamos por la RelativeLayout y sus propiedades.

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context=".MainActivity"

android:background="@drawable/background"
>
```

Un Layout es un contenedor y un RelativeLayout más concretamente es un contenedor que permite posicionar sus elementos respecto a la posición del propio Layout o de otros elementos.

Este RelativeLayout nos servirá de base para la interfaz y cubre toda la pantalla cosa que obtenemos con:

```
android:layout_width="match_parent"
android:layout_height="match_parent"
```

Para agregar padding o sangría a los lados usamos el atributo padding{lugar}.

El atributo tools:context se usa en la vista previa de la interfaz y no es relevante.

Finalmente con android:background agregamos la imagen de fondo que tenemos en la carpeta res/drawings-{resolución}

```
<LinearLayout
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:background="@drawable/linearlayout_bg"
android:orientation="vertical"
android:padding="10dp" >
```

LinearLayout es un Layout que dispone todos sus elementos uno debajo de otro o de izquierda a derecha según definamos sus propiedades. En este caso este Layout nos servirá como contenedor de los dos botones y pondrá los elementos uno debajo de otro porque establecemos "vertical" como valor de android:orientation.

Este Layout además hacer uso de un estilo que he definido en drawable:

```
<?xml version="1.0" encoding="utf-8"?>
<shape
xmlns:android="http://schemas.android.com/apk/res/android"
android:shape="rectangle">
 <solid android:color="#99000000" ></solid>
 <corners android:radius="8px"></corners>
 <strokeandroid:width="0dp" android:color="#A4C2E0"></stroke>
</shape>
```

Este estilo define un rectángulo de color gris muy oscuro, con los bordes redondeados. En este rectángulo se encontraran los botones.

De vuelta al archivo del Layout de calibración definimos los botones:

```
<Button
android:id="@+id/btnCalibrar"
style="@style/DefaultButtonText"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_margin="4dp"
android:background="@drawable/button_default_bg"
android:padding="10dp"
android:text="@string/boton" />

<Button
android:id="@+id/btnBorrar"
```

```
style="@style/DefaultButtonText"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_margin="4dp"
android:background="@drawable/button_default_bg"
android:padding="10dp"
android:text="@string/reset" />
```

android:id define el id de los botones, con el que en el código hemos obtenido sus referencias.

style define el estilo del botón.

android:layout_width y android:layout_height definen el ancho y el alto de los botones. Elegimos match parent para el ancho, de forma que ocupe de ancho todo el layout que lo contiene, pero para el alto elegimos wrap_content, que calcula un alto suficiente como para que el texto sea legible.

android:layout_margin="4dp" define un margen respecto a los límites del botón con el siguiente elemento.

android:padding="10dp" expande los límites del botón.

android:text define el texto por defecto del botón. En este caso los textos están almacenados en el archivo de /res/values/strigs.xml

android:background lo usamos para dar formato al botón. El archivo que contiene el formato se encuentra en /res/drawable y es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<selector
xmlns:android="http://schemas.android.com/apk/res/android">
<itemandroid:state_pressed="true" >
<shape>
<solid
android:color="#f8f9fa" />
<stroke
android:width="1dp"
android:color="#d2d2d2" />
<corners
android:radius="1dp" />

</shape>
</item>
<item>
```

```
<shape>
<gradient
android:startColor="#f8f9fa"
android:endColor="#d2d2d2"
android:angle="270" />
<stroke
android:width="1dp"
android:color="#d2d2d2" />
<corners
android:radius="1dp" />

</shape>
</item>
</selector>
```

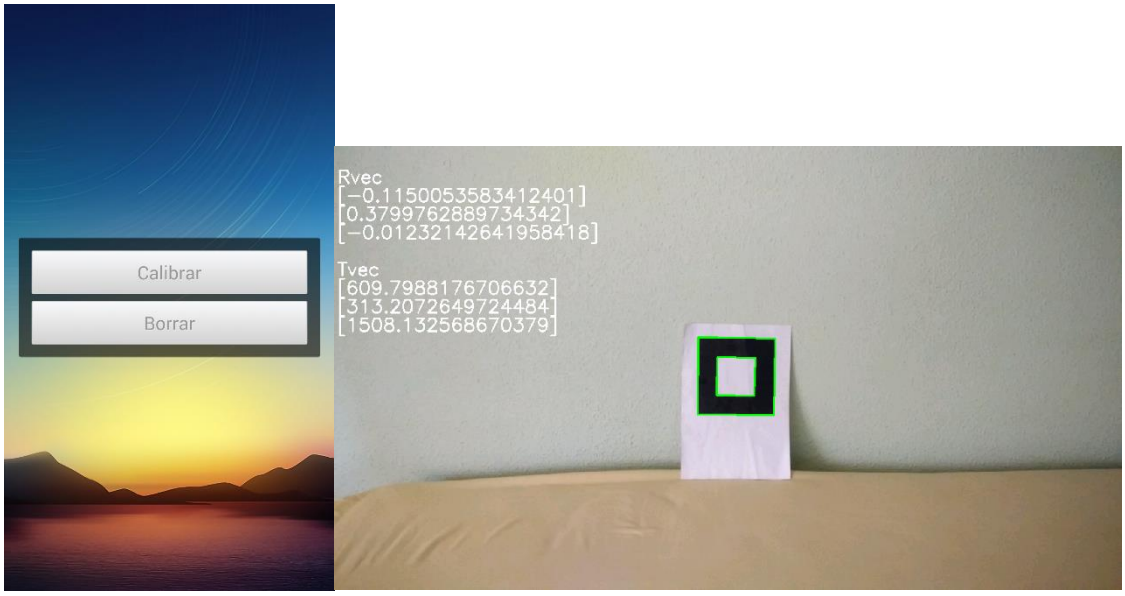
Ahora vamos a ver el fichero layout de la Activity de localización. Este archivo es mucho más sencillo dado que, como hemos visto en la fase de diseño, únicamente consta de una superficie donde se muestran los fotogramas capturados y procesados.

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent" >

<org.opencv.android.JavaCameraView
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:id="@+id/camara" />

</LinearLayout>
```

Esto es todo lo que necesitamos para definir la interfaz de nuestra segunda actividad.

Consta simplemente de un LinearLayout que nos sirve de contenedor para el elemento JavaCameraView que mostrará los fotogramas y que llena completamente el layout.

El resultado de la interfaz es:



Activity de calibración (izq) y Activity de localización (dcha.)

6.5 Archivos fuente Java/C++

En este apartado vamos a ver como desarrollar el código Java y C++ de la aplicación y que herramientas nos proporcionan tanto Android como OpenCv para facilitarnos el trabajo.

Empezamos definiendo el método onCreate de Android del módulo de calibración que se ejecuta al iniciarse una Actividad.

En este método queremos iniciar los parámetros de la Actividad, con lo cual cargamos el layout que contiene la interfaz visual de nuestra actividad y obtenemos una referencia al archivo XML con getSharedPreferences(). Este archivo contendrá, o contiene en caso de ser una ejecución posterior, los parámetros intrínsecos. Obtenemos también las referencias a los botones de la interfaz mediante las ids que declaramos en el layout. Además comprobamos si la cámara ya ha sido calibrada para establecer el texto que se muestra en btnCalib.

```
public void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);

	//Cargamos el layout
	setContentView(R.layout.activity_main);

	//Obtenemos la referencia al archivo de preferencias
	prefs=getSharedPreferences("Calibracion",Context.MODE_PRIVATE);

	//Creamos el directorio donde almacenaremos las fotografías de
```

```
        calibracion
        File                wallpaperDirectory                =                new
        File(Environment.getExternalStorageDirectory() + "/Calibracion");
        wallpaperDirectory.mkdirs();

        //Obtenemos una referencia a los controles de la interfaz
        final Button btnCalib = (Button)findViewById(R.id.BtnCalib);
        final Button btnReset = (Button)findViewById(R.id.BtnReset);
        if(isCalibrated())
              btnCalib.setText("Continuar");
```

Donde la función isCalibrated es una función auxiliar que comprueba si existe un parámetro de calibración para así saber si ya tenemos los parámetros y por lo tanto ya hemos calibrado la cámara en alguna sesión anterior de la aplicación. Para ello llama a la función contains(String key) de la clase SharedPreferences que comprueba si existe alguna entrada en las preferencias con la clave pasada como argumento. Está definida de la siguiente forma:

```
private boolean isCalibrated() {

      boolean contiene=prefs.contains("elem1");
      return contiene;
}
```

Ahora vamos a definir las funciones de cada botón asignándoles un objeto de la clase OnClickListener que tiene un método OnClick que se ejecutará cuando los pulsemos.

Primero vamos a definir el de btnReset que nos borrara los parámetros de calibración. Para esto obtenemos una referencia al editor de las preferencias y llamamos a la función clear() que borra las preferencias pero no el propio archivo. Hecho esto debemos confirmar los cambios con commit().

Hacemos esto para poder recalibrar la imagen, por ejemplo, en caso de que deseemos calibrarla con más fotografías o con otras diferentes. Finalmente cambiamos el texto que se muestra en btnCalib para que el usuario vea que los datos han sido borrados y es necesario volver a calibrar la cámara.

```
btnReset.setOnClickListener(new OnClickListener(){
      public void onClick(View v) {
            prefs.edit().clear().commit();
            btnCalib.setText("Calibrar");
      }
});
```

A continuación vamos a definir el OnClickListener de btnCalib que nos permitirá calibrar la cámara y acceder a la segunda Activity. Como contiene bastantes líneas vamos a verlo parte por parte.

Primero inicializamos las variables que contendrán los parámetros intrínsecos al finalizar la calibración. Estas variables deben ser matrices de 3X3 y de 8X1 para la matriz intrínseca y los coeficientes de distorsión respectivamente.

```
btnCalib.setOnClickListener(new OnClickListener() {
	@Override
	public void onClick(View v) {
		//Inicializamos los parametros
		Mat matriz=Mat.eye(3, 3, CvType.CV_64F);
		Mat coeffs=Mat.zeros(8,1,CvType.CV_64F);
```

A continuación comprobamos si ha sido calibrada ya la cámara. En caso negativo vamos a recopilar las rutas de las imágenes de calibración para posteriormente pasarlas al método de calibración definido en la librería nativa. Para ello debemos obtener la raíz del almacenamiento con getExternalStorageDirectory(), una función de la clase Environment que forma parte de la API de Android. Una vez tenemos la raíz exploramos la carpeta calibración y metemos las rutas en un array:

```
		//Si no esta calibrada la calibramos
		if(!isCalibrated()){
			btnCalib.setText("Calibrando");
			//Obtenemos las rutas de las imágenes de calibracion
			File root=Environment.getExternalStorageDirectory();
			String path=root.getAbsolutePath()+"/Calibracion";
			File pictures= new File(path);
			String lista1[]=pictures.list();
			String pathList[]= new String[lista1.length];
			for(int i=0;i<lista1.length;i++){
				pathList[i]=path+"/"+lista1[i];
			}
```

Una vez tenemos la lista de rutas de las imágenes llamamos a la función calibrate definida en la librería que hemos programado en C++. Como argumentos le debemos pasar la lista de rutas y las direcciones de los objetos Matriz nativos que contendrán los parámetros, cosa que hacemos con el método getNativeObjAddr() que implementa la clase Mat de Android OpenCv:

```
			//Llamamos a la funcion de calibracion
			calibrate(pathList,		matriz.getNativeObjAddr(),
				coeffs.getNativeObjAddr());
```

Esta función, a pesar de ser una función de una librería, por ser de una librería nativa debe tener definida una cabecera en la propia clase Java de la siguiente forma:

```
	private native void calibrate(Object[] path, long l, long m);
```

Vamos a ver ahora como esta implementado el método calibrate en la librería nativa que hemos programado en C++.

Para empezar las cabeceras de las funciones nativas son especiales y deben tener la estructura:

```
JNIEXPORT {tipo_devuelto} JNICALL Java_{paquete}_{Clase}_{nombre_funcion}(JNIEnv *env, jobject thisObj, [argumentos])
```

Donde {paquete} es el nombre del paquete con los puntos sustituidos por guiones bajos y {Clase} es el nombre de la clase que llama a la función

En este caso tiene la siguiente forma:

```
JNIEXPORT void JNICALL Java_upm_tfg_Calibracion_calibrate(JNIEnv *env, jobject thisObj, jobjectArray archivos, jlong matPtr, jlong coeffPtr){
```

Los argumentos que le pasamos en java son los que hemos definido: archivos, matPtr y coeffPtr.

Para empezar vamos a convertir las direcciones que hemos pasado como parámetros en objetos Mat en C++ que podamos manejar. Además declararemos algunas de las variables que usaremos en la función y obtendremos la longitud del array de rutas para luego procesarlo haciendo uso de la variable env que nos da información sobre los parámetros.

```
//Recuperamos los objetos y parametros
Mat& cameraMatrix=*(Mat*) matPtr;
Mat& distCoeffs=*(Mat*) coeffPtr;
int stringCount = env->GetArrayLength(archivos);

//Declaramos variables
vector<string> imageList;
vector<vector<Point2f> > imagePoints;
vector<vector<Point3f> > objectPoints(1);
vector<Point2f> pointBuf;
```

A continuación necesitamos convertir el array de java que hemos pasado como argumento a un vector de strings de C++. Nuevamente la variable env tiene funciones que nos permiten acceder a los elementos y reconvertirlos.

```
//Transformamos el array de rutas de java en un vector de strings de C++
for (int i=0; i<stringCount; i++) {
	jstring imgPath = (jstring) env->GetObjectArrayElement(archivos, i);
	const char *rawString = env->GetStringUTFChars(imgPath, 0);
	string img(rawString);
	imageList.push_back(img);
	env->ReleaseStringUTFChars(imgPath,rawString);
}
```

Ahora vamos a emplear la función imread(String File) de OpenCv para abrir las imágenes y leerlas como Matrices que podamos pasar como argumentos a la función findChessBoardCorners() que obtiene las coordenadas en la imagen que luego usaremos para calcular los parámetros intrínsecos en la calibración, es decir, nos provee los imagePoints. Esta función tiene limitaciones en relación al tamaño de la imagen. En imágenes muy grandes no es capaz de encontrar los puntos, por lo que antes de usarla debemos reescalar la imagen y hacerla más pequeña.

```
for(int i=0;i<(int)imageList.size();i++){
	bool found=false;
	//Leemos la imagen
	Mat view=imread(imageList[i]);
	Mat dst;

	//Reescalamos la imagen
	resize(view, dst, Size(), 0.3, 0.3, CV_INTER_AREA);

	//Hallamos los imagePoints y los guardamos en el vector
	found=findChessboardCorners(dst,Size(8,5),pointBuf,0);
	if(found){
		imagePoints.push_back(pointBuf);
	}
}
```

Una vez hemos obtenido los imagePoints debemos calcular los objectPoints. Para ello definimos una función que los calcule en base a nuestro patrón. En este caso los obtenemos de la siguiente manera.

```
objectPoints[0].clear();
getObjectPoints(objectPoints[0]);
objectPoints.resize(imagePoints.size(),objectPoints[0]);
```

La función getObjectPoints será:

```
void getObjectPoints(vector<Point3f>& corners)
{
	for( int i = 0; i < tamano.height; ++i )
		for( int j = 0; j < tamano.width; ++j )
			corners.push_back(Point3f(float(  j*chess_size  ),  float(
				i*chess_size ), 0));
}
```

Donde chess_size es el tamaño del lado de cada casilla, en este caso 29 mm.

Para terminar declaramos los vectores rvec y tvec y llamamos a la función calibrateCamera anteriormente expuesta. Esta función escribirá los resultados de los parámetros intrínsecos en la memoria que inicializamos en Java y de esta forma no tendremos que devolver ningún objeto.

```
vector<Mat> rvecs, tvecs;
double rms = calibrateCamera(objectPoints, imagePoints, Size(8,5),
      cameraMatrix, distCoeffs, rvecs, tvecs);
```

Ahora de vuelta a java lo que vamos a hacer es guardar los parámetros que hemos recibido de la calibración en el archive XML de preferencias e indicar que así ha sido cambiando el texto que se muestra en btnCalib. Para ello definimos dos funciones saveMatrix() y saveCoeffs():

```
//Guardamos los parametros de la matriz y los coeficientes
saveMatrix(matriz);
saveCoeffs(coeffs);
}
```

Debido a la forma en que se accede a los elementos de las matrices en Java y los tipos soportados por las preferencias el código para estas funciones es excesivamente voluminoso sin embargo es muy sencillo.

```
protected void saveCoeffs(Mat coeffs) {
      //Obtenemos la referencia al editor
      SharedPreferences.Editor editor= prefs.edit();

      //Accedemos a los elementos relevantes
      double[] elemen1=coeffs.get(0, 0);
      double[] elemen2=coeffs.get(1, 0);
      double[] elemen3=coeffs.get(2, 0);
      double[] elemen4=coeffs.get(3, 0);
      double[] elemen5=coeffs.get(4, 0);

      //Los convertimos a un tipo soportado por las preferencias
      float e1=(float) elemen1[0];
      float e2=(float) elemen2[0];
      float e3=(float) elemen3[0];
      float e4=(float) elemen4[0];
      float e5=(float) elemen5[0];

      //Los guardamos
      editor.putFloat("coef1", e1);
      editor.putFloat("coef2", e2);
      editor.putFloat("coef3", e3);
      editor.putFloat("coef4", e4);
      editor.putFloat("coef5", e5);
      editor.commit();
}

protected void saveMatrix(Mat matriz) {
      SharedPreferences.Editor editor= prefs.edit();
      double[] elemen1=matriz.get(0, 0);
      double[] elemen2=matriz.get(0, 2);
```

```
        double[] elemen3=matriz.get(1, 1);
        double[] elemen4=matriz.get(1, 2);
        float e1=(float) elemen1[0];
        float e2=(float) elemen2[0];
        float e3=(float) elemen3[0];
        float e4=(float) elemen4[0];
        editor.putFloat("elem1", e1);
        editor.putFloat("elem2", e2);
        editor.putFloat("elem3", e3);
        editor.putFloat("elem4", e4);
        editor.commit();
}
```

Finalmente llamamos a la segunda actividad que se encargará de localizar la imagen y mostrar los parámetros extrínsecos de los fotogramas al usuario. Para eso creamos un objeto Intent al que indicamos la clase que llama y la clase que es llamada para la actividad que vamos a lanzar.

```
//Creamos el Intent
Intent intent = new Intent(Calibracion.this, Localizacion.class);

//Iniciamos la nueva actividad
startActivity(intent);

}
```

Con esto hemos terminado el OnCreate, sin embargo todavía nos falta por revisar una parte fundamental del primer módulo. Dicha parte es la encargada de cargar las librerías de OpenCv puesto que es recomendable cargarlas a través de la aplicación OpenCv Manager en lugar de estáticamente.

Para hacerlo definimos un atributo de clase de la clase BaseLoaderCallback que permitirá recibir el callback de la aplicación OpenCv manager cuando esté conectada a nuestra aplicación y preparada para proveernos las librerías que requerimos. Además como necesitamos las librerías de OpenCv también para la parte nativa del sistema, debemos asegurarnos de que estamos conectados antes de cargar la librería nativa. Por esta razón vamos a hacer un override dentro del objeto BaseLoaderCallback de la función OnManagerConnected para que cuando esta se ejecute, es decir, cuando estemos conectados al manager carguemos la librería nativa.

```
private BaseLoaderCallbackmLoaderCallback = new BaseLoaderCallback(this) {
        @Override
        public void onManagerConnected(int status) {
                switch (status) {
                //Si nos conectamos exitosamente cargamos la libreria nativa
                case LoaderCallbackInterface.SUCCESS:
                {
                        System.loadLibrary("NativeLib");
                        } break;
```

```
                default:
                {
                        super.onManagerConnected(status);
                } break;
                }
        }
};
```

Una vez hecho esto debemos inicializar la comunicación del BaseLoaderCallback con el OpenCv Manager. Esto lo hacemos en el método onResume() que es un método que existe en todas las Actividades de Android y se ejecuta cuando una Actividad comienza pero después del onCreate(). Para iniciar la escucha del BaseLoaderCallback usamos el método initAsync() indicándole la versión del OpenCv la Actividad y la variable que contiene el BaseLoaderCallback para que sepa dónde dirigirse.

```
@Override
public void onResume()
{
        super.onResume();
        if  (!OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_9,  this,
                mLoaderCallback))
        {
                Log.e("Pruebas", "No se pudo conectar a OpenCv Manager");
        }
        else
                Log.e("Pruebas", "Conectado a OpenCV Manager");
}
```

Ya hemos terminado con el módulo de calibración y la primera Actividad de nuestra aplicación.

Ahora vamos a ver el módulo de localización. Este módulo, como el anterior, consta de una Actividad y llama a una función nativa definida en la librería en C++.

En este caso el onCreate es muy sencillo y consiste en llamar a la función addFlags() que nos permitirá mantener la pantalla encendida, puesto que no queremos que se apague ni tener que tocarla constantemente para que no ocurra, cargar el layout de la interfaz y activar la cámara. Además en este método cargaremos los parámetros intrínsecos en atributos de clase.

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        //Mantenemos la pantalla encendida
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);

        //Cargamos el layout
        setContentView(R.layout.camera);

        //Cargamos los parametros intrinsecos
```

```
        cameraMatrix=loadMatrix();
        distCoeffs=loadCoeffs();

        //Obtenemos la referencia a la camara y la activamos
        mOpenCvCameraView = (CameraBridgeViewBase) findViewById(R.id.camara);
        mOpenCvCameraView.setCvCameraViewListener(this);
}
```

Como se puede observar hay dos funciones definidas para cargar los parámetros intrínsecos. Estas funciones hacen uso del archivo de preferencias que hemos modificado en la Actividad de calibración y reconstruyen las matrices.

```
private Mat loadMatrix() {

        //Inicializamos la matriz y obtenemos la referencia a las preferencias
        Mat resultado=Mat.zeros(3,3,CvType.CV_64F);
        SharedPreferences                                                prefs=
              getSharedPreferences("Calibracion",Context.MODE_PRIVATE);

        //Obtenemos los parametros
        float aux[]= new float[4];
        aux[0]=prefs.getFloat("elem1", 1);
        aux[1]=prefs.getFloat("elem2", 1);
        aux[2]=prefs.getFloat("elem3", 1);
        aux[3]=prefs.getFloat("elem4", 1);

        //Colocamos los parametros en sus sitios
        resultado.put(0, 0, aux[0]);
        resultado.put(0, 2, aux[1]);
        resultado.put(1, 1, aux[2]);
        resultado.put(1, 2, aux[3]);
        resultado.put(2, 2, 1.0);
        return resultado;
}



private Mat loadCoeffs() {

        Mat resultado=Mat.zeros(5,1,CvType.CV_64F);
        SharedPreferences                                                prefs=
              getSharedPreferences("Calibracion",Context.MODE_PRIVATE);
        float aux[]= new float[5];
        aux[0]=prefs.getFloat("coef1", 1);
        aux[1]=prefs.getFloat("coef2", 1);
        aux[2]=prefs.getFloat("coef3", 1);
        aux[3]=prefs.getFloat("coef4", 1);
        aux[4]=prefs.getFloat("coef5", 1);
        resultado.put(0, 0, aux[0]);
        resultado.put(1, 0, aux[1]);
        resultado.put(2, 0, aux[2]);
        resultado.put(3, 0, aux[3]);
        resultado.put(4, 0, aux[4]);
```

```
	return resultado;
}
```

Ahora que tenemos los parámetros ya estamos preparados para recibir los fotogramas de la cámara y procesarlos. Para ello nuestra actividad implementara la interfaz de OpenCv CvCameraViewListener. Esta interfaz incluye el método onCameraFrame() que nos permitirá procesar cada frame que obtengamos. En este método llamaremos a la función findSquares que definimos en la librería nativa para obtener la posición del patrón de localización y por lo tanto los parámetros extrínsecos.

Para ello transformamos el fotograma, que es de tipo CvCameraViewFrame, en un objeto tipo Mat con la función rgba() que implementa CvCameraViewFrame. A continuación llamaremos a la función findSquares, pasándole como argumentos la matriz del fotograma y los parámetros intrínsecos. Como ya vimos en el módulo de calibración para pasar los objetos debemos usar la dirección del objeto nativo.

```
public Mat onCameraFrame(CvCameraViewFrame inputFrame) {
	//Transformamos el fotograma en un objeto tipo Mat
	mRgba = inputFrame.rgba();

	//Llamamos a la funcion definida en la libreria nativa
	findSquares(mRgba.getNativeObjAddr(),  cameraMatrix.getNativeObjAddr(),
		distCoeffs.getNativeObjAddr());

	return mRgba;
}
```

NOTA: Hay que recordar que debemos definir la cabecera de la función en java:

```
	private native void findSquares(long nativeObjAddr, long cM, long dC);
```

Ahora pasemos a la librería nativa para explicar el funcionamiento de la función findSquares detenidamente.

Ya vimos anteriormente que las cabeceras de las funciones nativas son diferentes a las funciones convencionales, por lo que la cabecera de la función findSquares con los argumentos mencionados seria así:

```
JNIEXPORT void JNICALL Java_upm_tfg_Localizacion_findSquares(JNIEnv *env,
jobject thisObj, jlong imagen,jlong cM, jlong dC){
```

Dentro de la función, como en calibrate(), lo primero que debemos hacer es construir los objetos a partir de las direcciones que pasamos como argumentos para poder manejarlos y declarar variables.

```
Mat& src= *(Mat*)imagen;
Mat& cameraMatrix= *(Mat*)cM;
Mat& distCoeffs= *(Mat*)dC;
```

```
vector<vector<Point> > contours, cuadrados, marca, ordenados;
vector<Vec4i> hierarchy;
Mat src_gray, canny_output, rvec, tvec, dst;
```

Ahora vamos a empezar a procesar el fotograma. Para hacerlo vamos a realizar un pre-procesamiento sobre la imagen. Primero vamos a transformarlo de formato RGB (imagen en color) a escala de grises (imagen en blanco y negro convencional). Después le aplicamos la función blur() que ayudara a mejorar los resultados cuando busquemos los contornos. Finalmente convertimos la imagen en escala de grises en una imagen binaria (solo tiene 2 colores, blanco o negro) que es el tipo de imagen que acepta la función que usaremos para detectar los contornos. En este último paso debemos definir primero un umbral que establecerá la frontera según la cual un pixel es 0 o 1 en la binarización.

```
//Transformacion de RGB a escala de grises
cvtColor( src, src_gray, CV_BGR2GRAY );

//Filtro
blur( src_gray, src_gray, Size(3,3) );

//Definicion del umbral y transformacion a binaria
int thresh=150;
Canny( src_gray, canny_output, thresh, thresh*2, 3 );
```

Una vez hemos pre-procesado la imagen vamos a hallar los contornos sobre la imagen binarizada. Para hallarlos usamos la función findContours() de OpenCv que nos devolverá una colección de todos los contornos, punto por punto, que detecte en la imagen.

```
findContours( canny_output, contours, hierarchy, CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );
```

Donde canny_output es la imagen binarizada, contours es el vector contornos encontrados, hierarchy es la jerarquía de los contornos, CV_RETR_TREE es un flag que define que se devuelvan todos los contornos y la jerarquía completa, CV_CHAIN_APPROX_SIMPLE es el método usado en la aproximación y Point(0,0) el offset o desplazamiento de las coordenadas sobre la imagen, en este caso ninguno.

Ahora tenemos una colección de contornos en contours que debemos filtrar pues solo nos interesa nuestro patrón.

Como filtrar contornos puede resultar complejo en este caso defino dos funciones, filtraVertices() y eliminateDoubleContours().

```
cuadrados=filtraVertices(contours);
marca=eliminateDoubleContours(cuadrados);
```

La primera se encarga de aproximar los vértices de los contornos y a partir de ahí eliminar todos aquellos que no tengan 4 vértices o no sean convexos, es decir, que no sean cerrados. Además elimina aquellos contornos que sean muy pequeños. Como resultado obtenemos una nueva lista de contornos, todos ellos definidos por 4 vértices y todos ellos convexos. Para obtener este resultado nos apoyamos en las funciones de OpenCv, approxPolyDP(), que aproxima los vértices; arcLength(), que calcula el perímetro de un área convexa o la longitud de una curva; isContourConvex(), que comprueba que un contorno sea convexo y contourArea(), que calcula el área de un contorno.

```
vector<vector<Point> > filtraVertices(vector<vector<Point> > contornos){
	vector<vector<Point> > cuadrangulos;
	cuadrangulos.clear();
	vector<Point> aprox;
	Point punto;
	for(int i=0; i < contornos.size(); i++){
		approxPolyDP(Mat(contornos[i]),                                aprox,
		    arcLength(Mat(contornos[i]), true)*0.02, true);

		if(aprox.size()==4	&&	isContourConvex(aprox)	&&
		    (contourArea(aprox)>200)){
		      cuadrangulos.push_back(aprox);
		}
	}
	return cuadrangulos;
}
```

La segunda función filtra los contornos con 4 vértices para encontrar la marca final. Para hacer esto se asegura de que el cuadrado externo y el cuadrado interno tengan áreas suficientemente diferentes, por lo menos un 10% menor (calculadas con contourArea()), dado que findcontours en ocasiones duplica los contornos (uno de los vértices es ligeramente diferente). Con esto no basta puesto que puede haber otros contornos que tengan un área menor en la imagen, por lo que también debemos comprobar que el contorno con el área más pequeña se encuentra dentro del más grande para lo cual usamos la función pointPolygonTest() que comprueba que un punto o vértice este dentro de un contorno.

```
vector<vector<Point>	>	eliminateDoubleContours(vector<vector<Point>	>
cuadrados){
	vector<vector<Point> > ncuadrados;
	int encontrado=0;
	for(int i=0; i<cuadrados.size();i++){
		for(int j=0; j<cuadrados.size() && encontrado==0;j++){
			if(i!=j){
				double area1=contourArea(cuadrados[i]);
				double area2=contourArea(cuadrados[j]);
				double threshold=area1/10.0;
				double	inside=	pointPolygonTest(cuadrados[i],
				    cuadrados[j][0], false);
```

```
                        if(area2<area1-threshold && inside>0){
                              ncuadrados.push_back(cuadrados[i]);
                              ncuadrados.push_back(cuadrados[j]);
                              encontrado=1;
                        }
                  }
            }
      }
      return ncuadrados;
}
```

Ahora si hemos localizado el patrón en forma de dos contornos, el cuadrado externo y el cuadrado interno. Por lo que necesitamos calcular los objectPoints, es decir, las coordenadas de los vértices del patrón de localización en el mundo real.

Como se estableció en la fase de diseño el patrón consiste en dos cuadrados concéntricos, el externo tiene lados de 200mm y el interno de 100mm. Por lo tanto la función que nos devuelve los objectPoints será:

```
vector<Point3f> getObjectPoints(){
      vector<Point3f> objectPoints;
      Point3f *puntos=new Point3f[8];
      puntos[0].x=0;puntos[0].y=0;puntos[0].z=0;
      puntos[1].x=200;puntos[1].y=0;puntos[1].z=0;
      puntos[2].x=0;puntos[2].y=200;puntos[2].z=0;
      puntos[3].x=200;puntos[3].y=200;puntos[3].z=0;
      puntos[4].x=50;puntos[4].y=50;puntos[4].z=0;
      puntos[5].x=150;puntos[5].y=50;puntos[5].z=0;
      puntos[6].x=50;puntos[6].y=150;puntos[6].z=0;
      puntos[7].x=150;puntos[7].y=150;puntos[7].z=0;
      for(int i=0;i<8;i++){
            objectPoints.push_back(puntos[i]);
      }

      return objectPoints;
}
```

Ahora ya tenemos los vértices de la marca en la imagen y en el mundo real, sin embargo no están en el mismo formato. Nuestro vector de objectPoints es correcto puesto que tiene los 8 vértices seguidos y en un orden concreto, pero para que la función solvePnP() pueda resolver los parámetros extrínsecos necesitamos que los dos vectores de puntos que representan los contornos de los cuadrados concéntricos en la imagen se encuentren en un solo vector, y que además estén en el mismo orden que los vértices de objectPoints.

En este trabajo he elegido ordenar los vértices de arriba abajo y de izquierda a derecha, primero los del cuadrado externo y después los del interno. Para ordenar los vértices vamos a definir una función que ordene cada contorno por separado y que obtenga primero los dos vértices con menor coordenada y, es decir, los dos superiores. De entre

estos dos vértices el que menor coordenada x tenga será considerado el primer vértice y el otro el segundo. De los dos vértices restantes el que menor coordenada x tenga será considerado el tercer vértice y finalmente el que resta será el cuarto.

```
vector<Point> sortCoord(vector<Point> contorno){
	vector<Point> ordenado;
	Point primero=contorno[0];
	Point segundo;
	Point tercero;
	int indice=0;

	//Vertice de menor coordenada y
	for(int i=1;i<contorno.size();i++){
		if(contorno[i].y<primero.y){
			primero=contorno[i];
			indice=i;
		}
	}
	contorno.erase(contorno.begin()+indice);

	//Segundo vertice de menor coordenada y
	segundo=contorno[0];
	indice=0;
	for(int i=0;i<contorno.size();i++){
		if(contorno[i].y<segundo.y){
			segundo=contorno[i];
			indice=i;
		}
	}
	contorno.erase(contorno.begin()+indice);

	//Decidimos el primer y segundo vertice
	if(primero.x<segundo.x){
		ordenado.push_back(primero);
		ordenado.push_back(segundo);
	}
	else{
		ordenado.push_back(segundo);
		ordenado.push_back(primero);
	}

	//Decidimos el tercer y cuarto vertice
	if(contorno[0].x<contorno[1].x){
		ordenado.push_back(contorno[0]);
		ordenado.push_back(contorno[1]);
	}
	else{
		ordenado.push_back(contorno[1]);
		ordenado.push_back(contorno[0]);
	}

	for(int i=0;i<ordenado.size();i++){
		//printPoint(ordenado[i]);
	}
```

```
	return ordenado;
}
```

Una vez definidas las funciones que nos proveen y ordenan los objectPoints y los imagePoints volvemos a la función principal findSquares donde vamos a llamar a solvePnP con los argumentos que hemos calculado, no sin antes instanciar los vectores de parámetros extrínsecos que obtendremos como resultado.

```
if(!marca.empty()){
	//printContour(marca[0]);
	//printContour(marca[1]);
	//Obtenemos los objectPoints
	vector<Point3f> objectPoints=getObjectPoints();

	//Ordenamos los imagePoints y los juntamos en un solo vector
	ordenados.push_back(sortCoord(marca[0]));
	ordenados.push_back(sortCoord(marca[1]));
	//printContour(ordenados[0]);
	vector<Point2f> imagePoints;
	for(int i=0;i<ordenados[0].size();i++){
		imagePoints.push_back(ordenados[0][i]);
	}
	for(int i=0;i<ordenados[1].size();i++){
		imagePoints.push_back(ordenados[1][i]);
	}

	//Instanciamos rvec y tvec y llamamos a solvePnP
	rvec=Mat::zeros(3, 1, CV_64F);
	tvec=Mat::zeros(3, 1, CV_64F);

	solvePnP(objectPoints,imagePoints,cameraMatrix,distCoeffs,rvec,tvec);
```

solvePnP() calculará los parametros extrínsecos que son el resultado final del procesamiento del fotograma. Por lo tanto lo único que debemos hacer ahora es dibujar el patrón en la pantalla y mostrar los datos. Para ello vamos a usar las funciones drawContours y putText de OpenCv que dibujan el contorno y ponen texto en una imagen respectivamente.

```
	drawContours(src, marca, -1, Scalar(0,255,0),2,8);
	putVector(src,"Rvec",rvec,0);
	putVector(src,"Tvec",tvec,150);
}
```

Donde la función putVector() es:

```
void putVector(Mat imagen,string mensaje,Mat vector,int flag){
	stringstream str1,str2,str3;
	str1<< vector.row(0);
	str2<< vector.row(1);
	str3<< vector.row(2);
```

```
        putText(imagen,  mensaje,  Point(5,60+flag),  FONT_HERSHEY_SIMPLEX,  1,
             Scalar::all(255), 1.5);
        putText(imagen, str1.str(), Point(5,90+flag), FONT_HERSHEY_SIMPLEX, 1,
             Scalar::all(255),1.5);
        putText(imagen,str2.str(), Point(5,120+flag), FONT_HERSHEY_SIMPLEX, 1,
             Scalar::all(255), 1.5);
        putText(imagen,str3.str(), Point(5,150+flag), FONT_HERSHEY_SIMPLEX, 1,
             Scalar::all(255), 1.5);
}
```

Teniendo la imagen modificada podemos volvemos a java donde lo único que queda por hacer es devolver la matriz de la imagen en la función onCameraFrame().

```
public Mat onCameraFrame(CvCameraViewFrame inputFrame) {
        //Transformamos el fotograma en un objeto tipo Mat
        mRgba = inputFrame.rgba();

        //Llamamos a la funcion definida en la libreria nativa
        findSquares(mRgba.getNativeObjAddr(),  cameraMatrix.getNativeObjAddr(),
              distCoeffs.getNativeObjAddr());

        return mRgba;
}
```

Igual que en el módulo de calibración hay que tener en cuenta que debemos conectarnos al OpenCv Manager y cargar la librería nativa con lo que debemos añadir el mismo código que antes. Aunque en este caso es más importante, puesto que la cámara no puede iniciarse sin el OpenCv Manager conectado correctamente.

```
private BaseLoaderCallbackmLoaderCallback = new BaseLoaderCallback(this) {
        @Override
        public void onManagerConnected(int status) {
              switch (status) {
              case LoaderCallbackInterface.SUCCESS:
              {
                    System.loadLibrary("NativeLib");
                    mOpenCvCameraView.enableView();
              } break;
              default:
              {
                    super.onManagerConnected(status);
              } break;
              }
        }
};

@Override
public void onResume()
{
        super.onResume();
        OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_3,         this,
        mLoaderCallback);
```

```
}
```

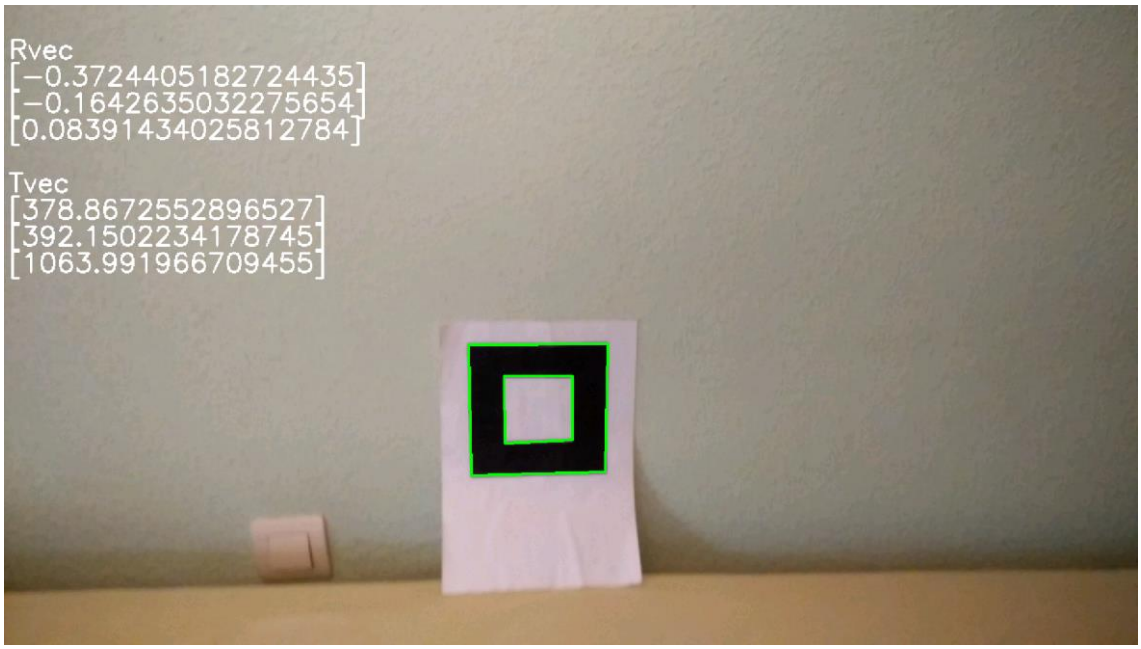
Finalmente definimos los métodos onPause(),onDestroy(), onCameraViewStarted() y onCameraViewStopped(). Los dos primeros métodos simplemente deshabilitan la cámara cuando ocurren los eventos de pausa o destrucción de la actividad. onCameraViewStopped() libera la memoria de la matriz del fotograma cuando la cámara se para y onCameraView() started inicializa la variable del fotograma.

```
public void onPause()
{
	super.onPause();
	if (mOpenCvCameraView != null)
		mOpenCvCameraView.disableView();
}

public void onDestroy() {
	super.onDestroy();
	if (mOpenCvCameraView != null)
		mOpenCvCameraView.disableView();
}

public void onCameraViewStopped() {
	mRgba.release();
}

public void onCameraViewStarted(int width, int height) {
	mRgba = new Mat(height, width, CvType.CV_8UC4);
}
```

7. EXPERIMENTOS

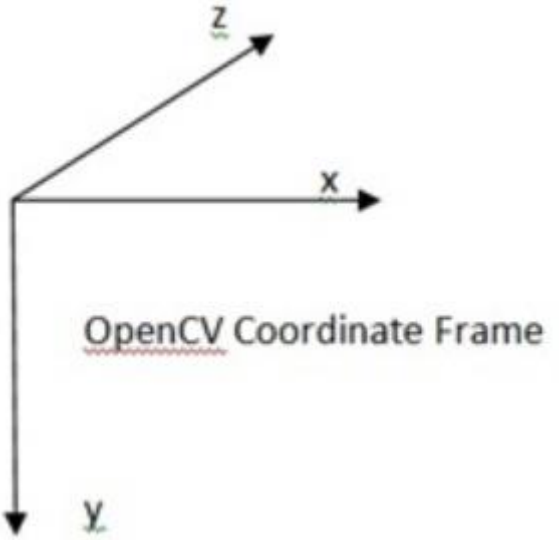
Para este apartado he realizado capturas de pantalla y una demo en video que se entregara con el propio trabajo.

Para entender los datos que nos aporta la aplicación debemos saber cómo funciona OpenCv. Veamos una imagen de muestra, resultado de la aplicación.



Como se puede observar en la esquina superior izquierda tenemos los datos que nos provee la aplicación. Las primeras cuatro líneas se corresponden con el vector de rotación, es decir la rotación del patrón de localización respecto de la cámara. Las siguientes cuatro líneas nos dan información sobre la traslación sobre la imagen respecto a la posición de la cámara, lo cual significa que representan el vector de traslación.

Para entender el vector de rotación hay que considerar los ejes que maneja OpenCv que tienen la siguiente estructura:



Como las coordenadas se muestran en pantalla (x,y,z) de arriba abajo, el primer número del vector de rotación representa el giro sobre el eje X, el segundo lo representa sobre el eje Y y el tercero sobre el Z. Los giros pueden tener valores tanto positivos como negativos dependiendo de la dirección del giro.

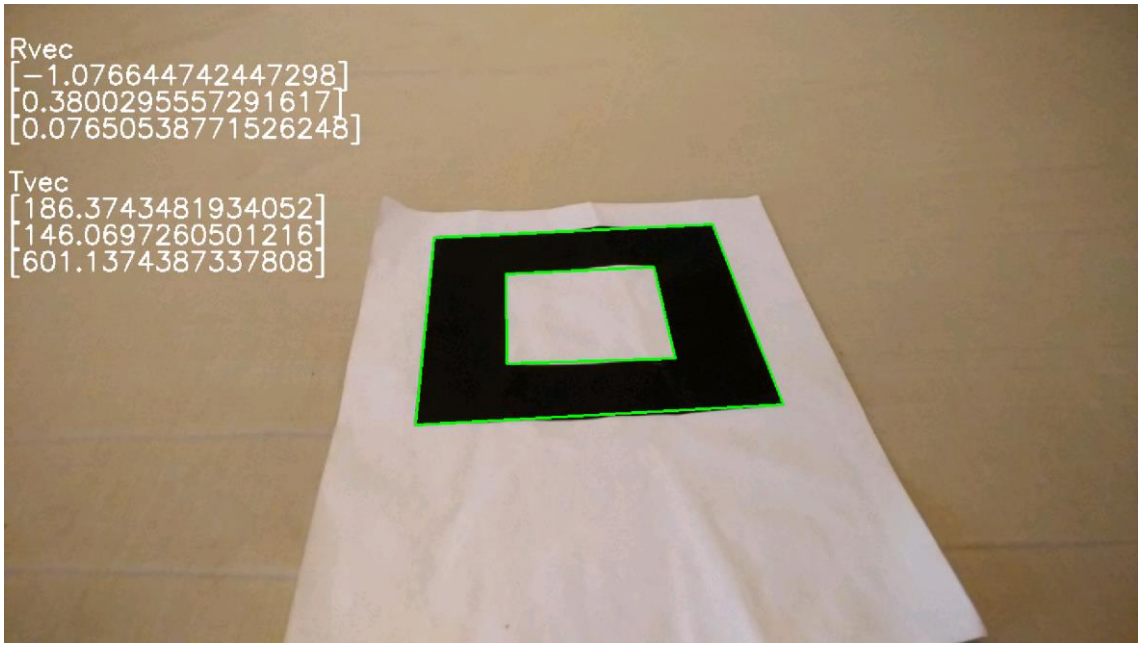
Respecto al vector de traslación OpenCv establece el origen de coordenadas en la esquina superior izquierda y como se puede apreciar el esquema todas las coordenadas de traslación de un objeto que se encuentre delante de la cámara deberían ser positivas dado que para que fueran negativas el objeto tendría que salir de la imagen por la izquierda (x negativa), por arriba (y negativa) o estar detrás de la cámara (z negativa)

Las coordenadas de traslación tienen como punto de referencia el primer punto del patrón que en este caso está establecido como el vértice superior izquierdo del cuadrado externo.

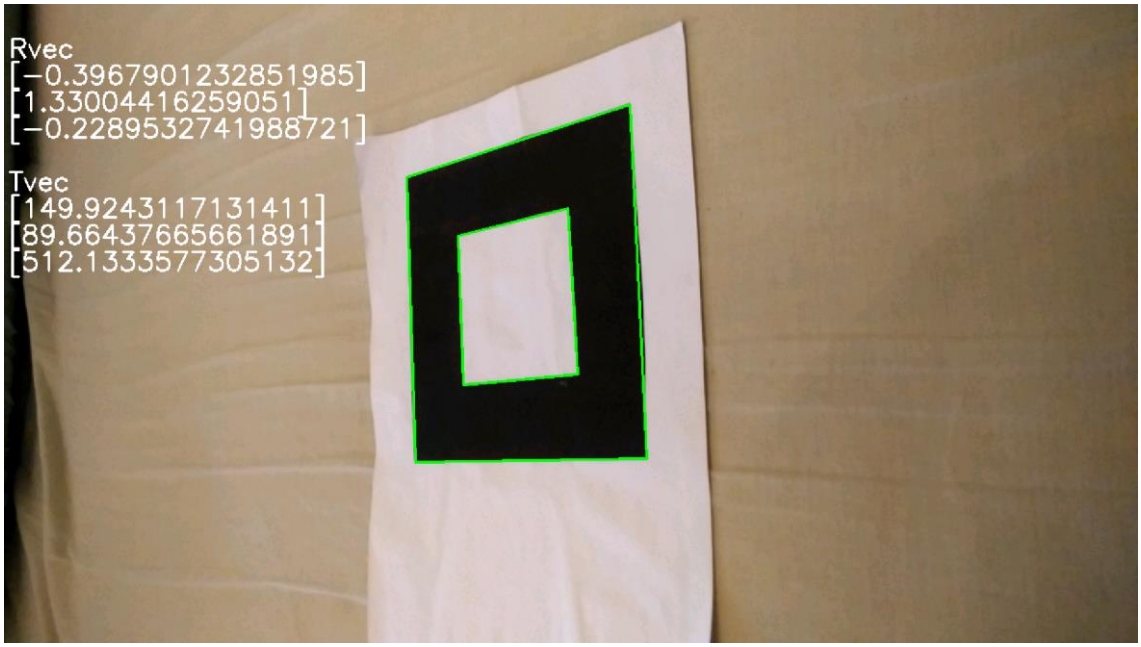
Debido al diseño del patrón de imposible determinar cuál es el lado inferior y cual es superior, dado que no hay ninguna referencia. Esto conlleva que al calcular la rotación en el eje Z el rango de giro vaya de -pi/4 a pi/4 dado que al superar ese ángulo de rotación el vértice opuesto al que desciende pasa a ser uno de los dos vértices superiores y el que desciende es considerado un vértice inferior. Sin embargo el giro en los ejes X e Y tiene un rango de –pi/2 a pi/2.

Ahora que podemos interpretar las coordenadas vamos a ver ejemplos:

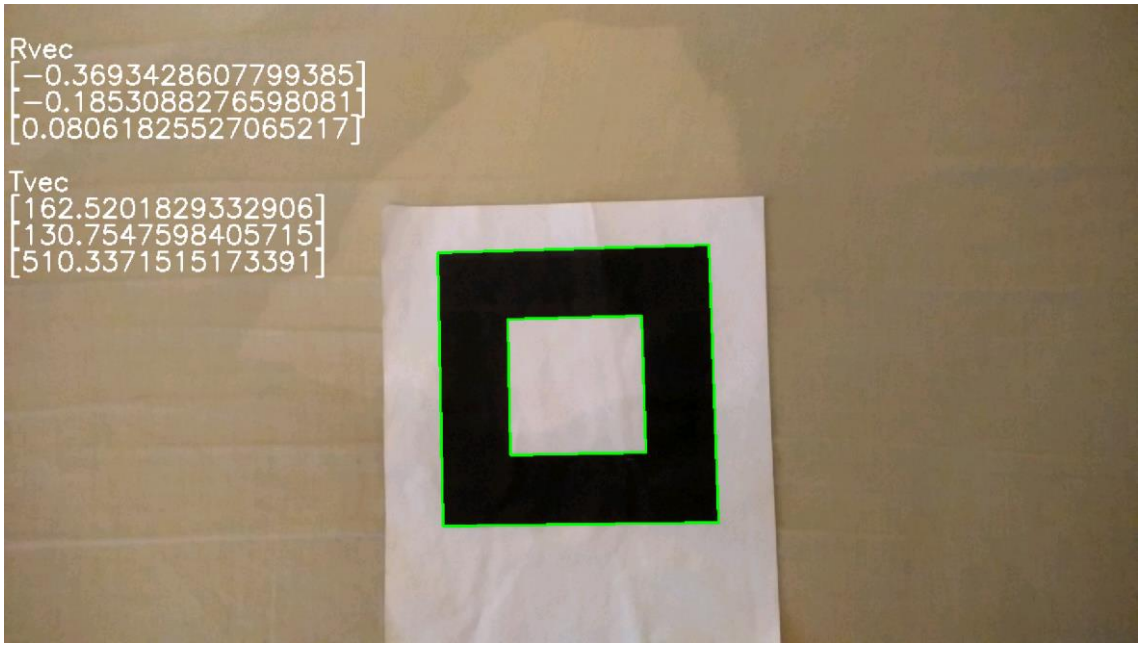
Rotación negativa sobre el eje X (-1.07 radianes):



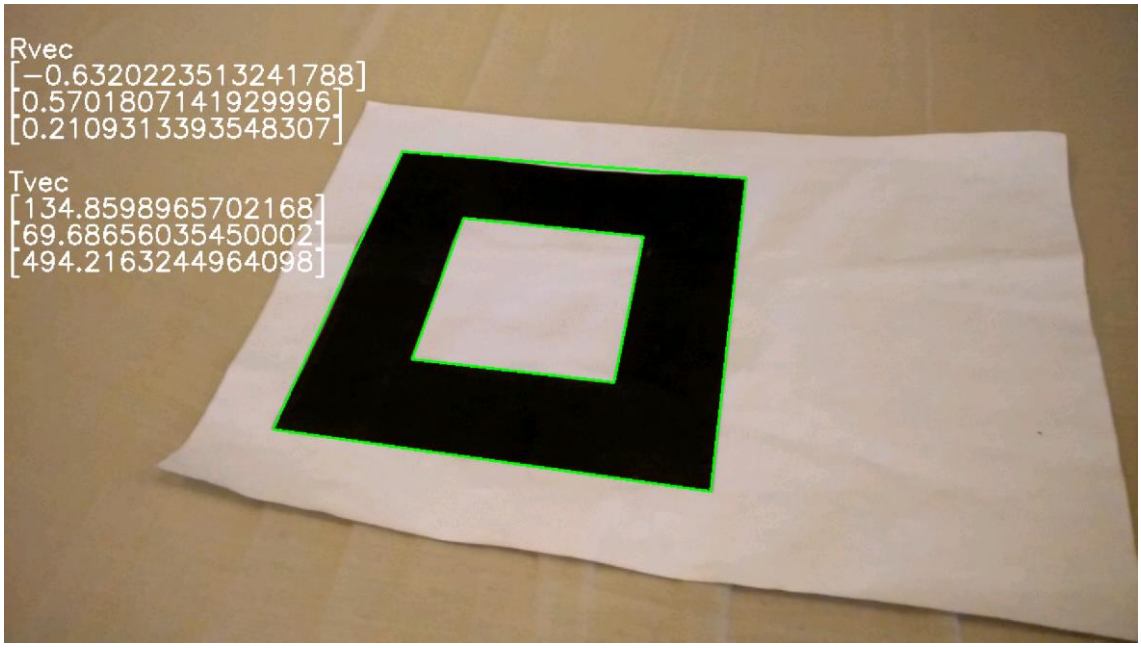
Rotación positiva sobre el eje Y (1.33 radianes):



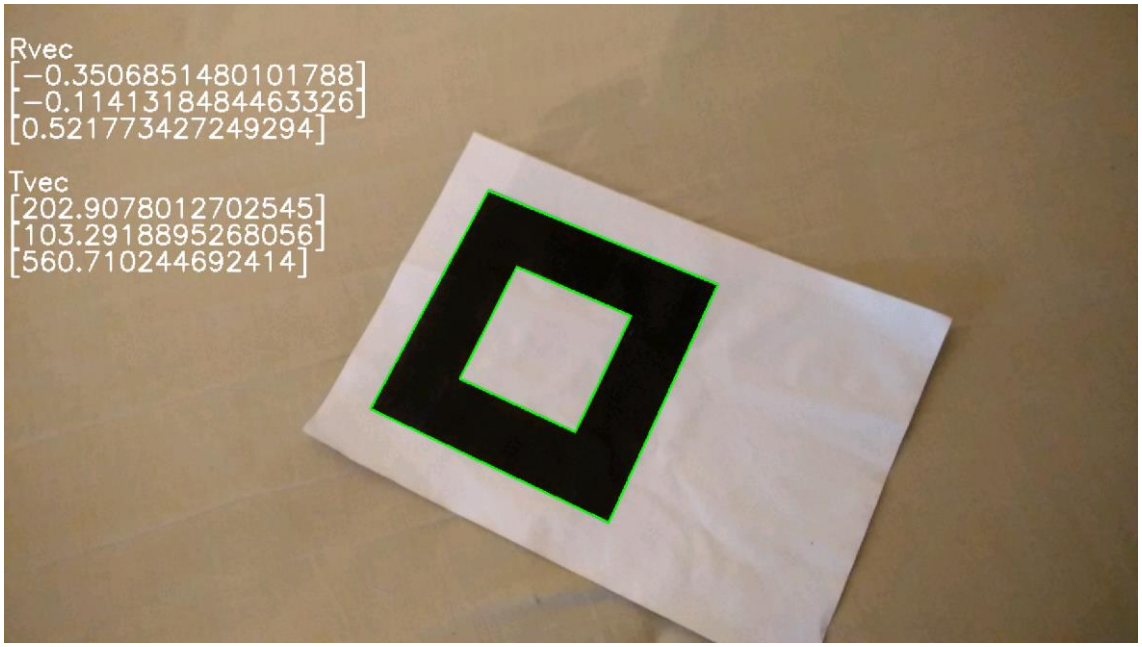
Casi ninguna rotación



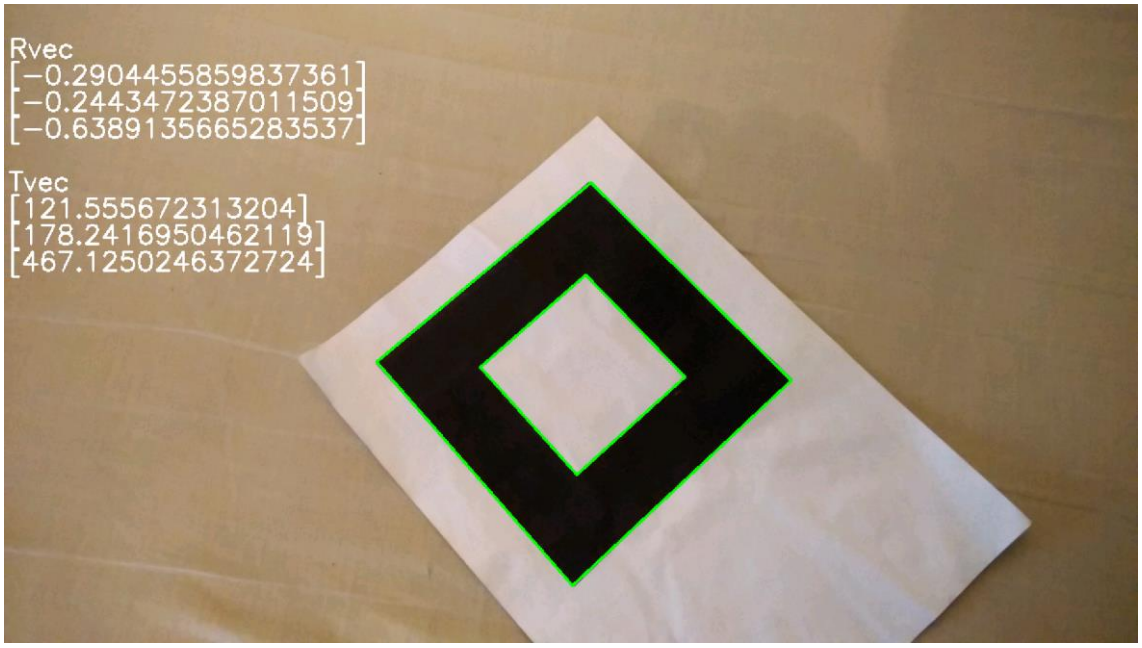
Mezcla de rotaciones:



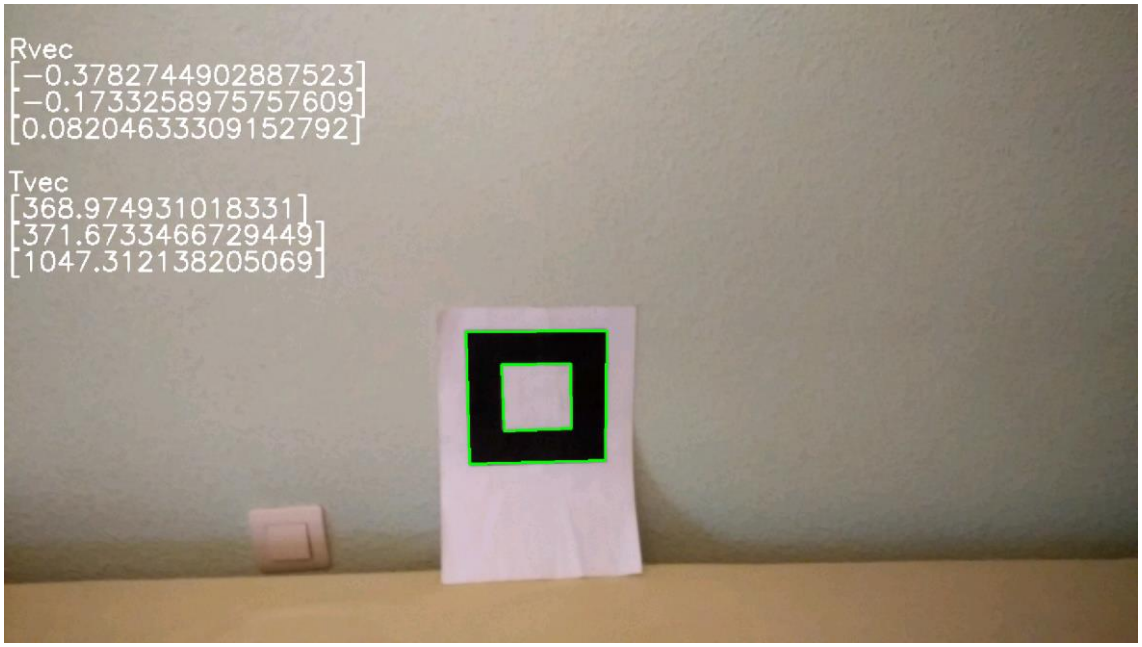
Rotación positiva en el eje Z (0.52 radianes):



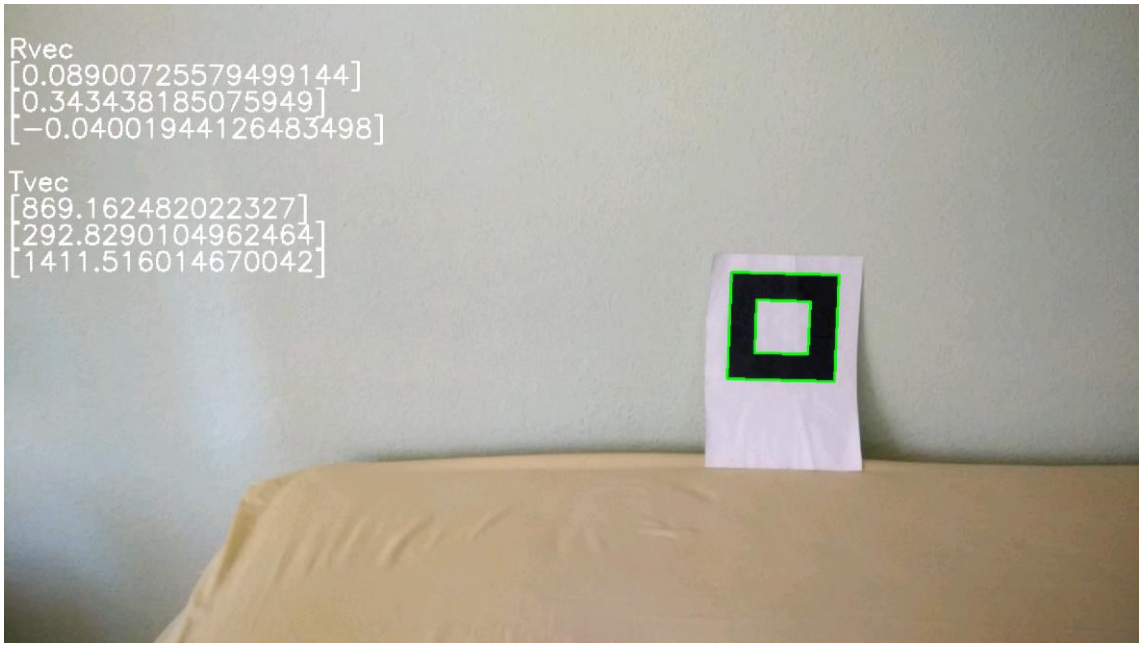
Rotación negativa en el eje Z (-0,63 radianes):



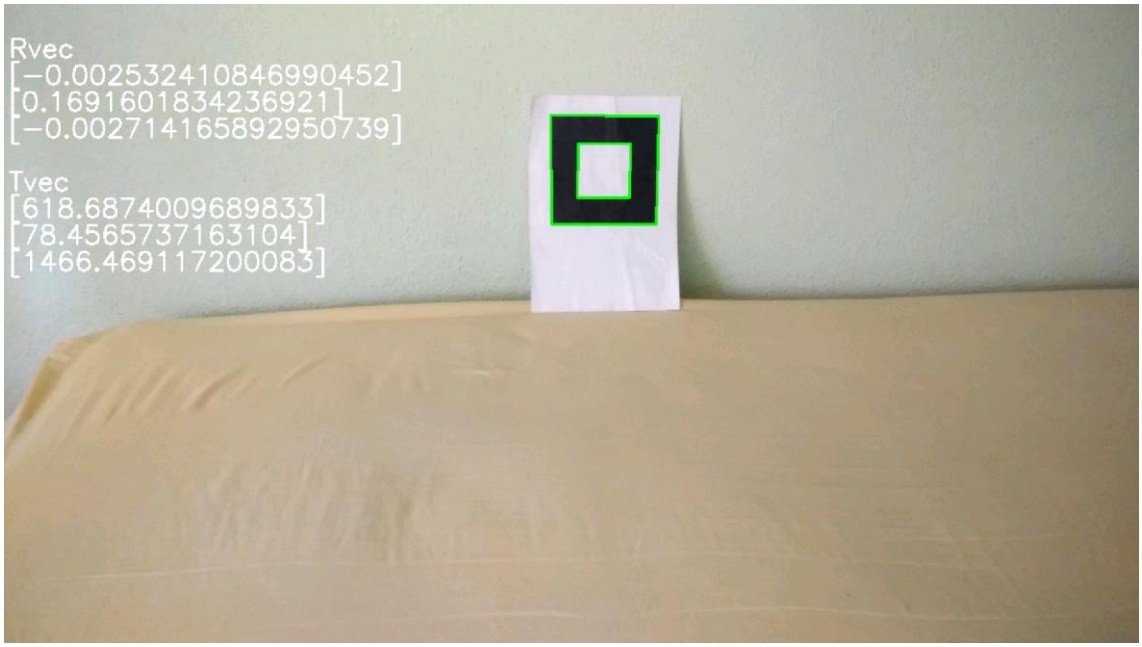
Traslación (alejamiento) en el eje Z:



Traslación en el eje X:



Traslación en el eje Y:



8. MANUAL DE USUARIO

En este capítulo describo los pasos para instalar el entorno de desarrollo, configurar el proyecto, compilarlo y ejecutarlo.

8.1 Instalación de Opencv Manager

Antes de usar la aplicación debemos instalar el OpenCv Manager para que la aplicación pueda acceder a las librerías de OpenCv conectándose a este.

Instalar el OpenCv Manager es muy sencillo. OpenCv Manager está disponible en la Play Store. Si no puedes acceder a la Play Store lo puedes encontrar en el siguiente link.

https://play.google.com/store/apps/details?id=org.opencv.engine&hl=es

Si no dispones de conexión a internet el apk de la aplicación está incluido en Android OpenCv en la ruta:

/OpenCv-2.4.9-android-sdk/apk/OpenCV_2.4.9_Manager_2.18_armv7a-neon.apk.

8.2 Instalar la aplicación

Para instalar la aplicación debemos activar la depuración USB[]. Para hacerlo seguimos los siguientes pasos (1 y 2 solo para Android 4.2 o superior) :

1. Nos dirigimos a **Ajustes->Informacion del Telefono** y nos desplazamos hasta el final
2. Pulsamos 7 veces sobre **Número de Compilación** y nos aparecerá un mensaje anunciando que ya somos un desarrollador.
3. Volvemos al menú anterior y debería aparecer una nueva opción “Opciones de desarrollo”. Entramos y marcamos la opción “Depuracion USB”.
4. Ahora conectamos el dispositivo al PC mediante el USB. Esperamos a que el ordenador reconozca el dispositivo correctamente
5. Abrimos una ventana de comandos y navegamos hasta {sdk de Android}/sdk/platform-tools.
6. Ejecutamos:

```
adb install {ruta_al_.apk}
```

Hecho esto ya deberíamos tener la aplicación instalada en nuestro dispositivo

8.3 Calibrar por primera vez

La primera vez que calibramos tenemos dos opciones:

1. Abrir la aplicación para que cree automáticamente la carpeta Calibracion
2. Crear la carpeta Calibracion mediante una aplicación o desde el PC

Una vez hecho esto seguimos los siguientes pasos:

1. Realizamos las fotografías de calibración del patrón de calibración desde diferentes ángulos y con buena iluminación. El patrón debe verse completo. Es recomendable hacer al menos 6-8 fotografías.
2. Introducimos las fotografías en la carpeta Calibración en la raíz del almacenamiento.
3. Abrimos la aplicación y pulsamos "Calibrar". Este proceso puede tardar un tiempo. Al final nos llevara automáticamente a la segunda Activity, la de localización.

Al finalizar el último paso la aplicación ya habrá guardado los parámetros de calibración en el dispositivo.

8.4 Recalibrar

1. Abrir la aplicación y pulsar en "Borrar".
2. Introducir o eliminar fotografías de la carpeta Calibración.
3. Abrir la aplicación nuevamente y pulsar "Calibrar".

8.5 Uso de la localización en sesiones posteriores de la aplicación

Para hacer uso de la localización del patrón en sesiones posteriores de la aplicación, una vez ha sido calibrada solo debemos abrir la aplicación y pulsar en "Continuar"

9. CONCLUSIONES

En este capítulo presento las conclusiones que he sacado del trabajo, así como lo que he aprendido.

Las primeras conclusiones me gustaría dedicarlas al desarrollo nativo (C/C++) en Android.

Creo que, aunque es una herramienta interesante y probablemente muy útil, actualmente el soporte que hay es bastante desastroso. En algunas versiones de eclipse no es posible realizar correctamente el indexado y por lo tanto aparecen errores de compilación y warnings falsos, dado que el programa puede compilar perfectamente.

Con la introducción de Android Studio puede que se solventen los problemas sobre el indexado al estar enfocado únicamente al desarrollo de aplicaciones. Sin embargo esta introducción se produjo tarde en el semestre por lo que no he podido explorar esta opción más a fondo.

Otro aspecto más relacionado con este trabajo es que toda la funcionalidad desarrollada en C++ podría haber sido desarrollada en Java puesto que Android OpenCv la soporta para Java. En caso de haberlo hecho el sistema habría sido sensiblemente más fácil de construir, incluida la configuración del entorno, aunque probablemente hubiese cierto coste computacional que no sabría determinar. Si el coste computacional fuese relativamente bajo yo recomendaría el desarrollo en Java en pro de la simplicidad.

Sobre Android en general he de decir que en el poco contacto que había tenido anteriormente había tenido la sensación de que era bastante complejo y antiintuitivo. Sin embargo, en este trabajo no he tenido ningún problema con Android y considero que tiene una API bastante completa, en tanto a lo que a este trabajo se refiere no puedo estar más satisfecho.

Considero que es importante comprender las partes de un proyecto en Android antes de comenzarlo, así como para qué sirven los archivos de “values” o cómo funciona el Android Manifest. Una vez comprendido nos podemos olvidar de los “problemas” que nos puedan dar esos archivos que en esencia son sencillos, aunque inusuales, y concentrarnos en desarrollar una aplicación de calidad.

En el tema de visión por computador y los conceptos de este campo que atañen al trabajo no tengo demasiado que decir. Dado que ya había hecho una práctica similar estaba familiarizado con los conceptos. He de confesar que era escéptico respecto al rendimiento de una aplicación de este tipo en un dispositivo móvil que tiene relativamente poca potencia. Sin embargo, los resultados han sido bastante buenos.

Me gustaría hacer hincapié en lo difícil que me ha resultado configurar el entorno y comenzar el trabajo y en el tiempo que requiere. No tengo duda alguna de que si hubiese sido más sencillo, sobretodo eliminando el desarrollo nativo, habría tenido más tiempo para desarrollar la aplicación y pulir los muchos detalles que quedan por revisar.

Sin embargo, y dado que es un trabajo y no un producto comercial, creo que los detalles pueden quedar al margen a cambio de los conocimientos que he adquirido.

Sin duda alguna mis conocimientos de Android han aumentado considerablemente, así como del desarrollo nativo el cual no conocía en un principio. Además, este trabajo ha refrescado mis conocimientos de las técnicas de visión por computador y los ha profundizado gracias a la búsqueda de información que he llevado a cabo para escribir el capítulo de introducción a la visión por computador.

En general estoy contento con el trabajo y con los conocimientos que he aprendido y aplicado y sin duda me habría gustado dedicar más tiempo a hacer una interfaz más bonita, un código más limpio o una aplicación más completa.

Como ultima conclusión, y dado que este es un trabajo enfocado a ser realizado por futuros alumnos en alguna asignatura, diría que es un trabajo muy adecuado si es desarrollado en Java. Si los alumnos intentaran desarrollarlo en C/C++, como se ha hecho en este trabajo, podrían encontrarse faltos de tiempo y de conocimientos y tener que dedicar una cantidad de tiempo excesiva. Suponiendo que el objetivo de la práctica es familiarizarse con los conceptos de visión por computador que se exponen, el desarrollo nativo es innecesario. Y si la mejora de rendimiento que supone no es significativa en el funcionamiento de la aplicación, no veo ninguna razón para imponerlo.

9.1 Posibles mejoras

En este apartado voy a detallar una lista de mejoras para el sistema dado que no es ni de cerca perfecto.

- Herramienta de debugging: Herramienta para mostrar o no los parámetros y el patrón de localización dibujados en pantalla.
- Cámara integrada en el módulo de calibración + comprobación en tiempo real de que las fotografías son correctas.
- Explorador de la carpeta de calibración integrado en la aplicación.
- SeekBar para seleccionar el umbral de sensibilidad en el módulo de localización
- Soporte para diversas resoluciones + modificaciones automáticas de los parámetros intrínsecos cuando es necesario.
- Soporte para formato 4:3 y sus resoluciones.
- Proyección de elementos sobre el patrón.

BIBLIOGRAFIA

Wikipedia sobre entornos de desarrollo. Disponible en:
http://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado

Wikipedia sobre visión por computador. Disponible en:
http://en.wikipedia.org/wiki/Computer_vision

Documentación sobre la teoría de la calibración. Disponible en:
http://docs.opencv.org/doc/tutorials/calib3d/camera_calibration/camera_calibration.html

Más teoría sobre la calibración. Disponible en:
http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

Documentación sobre Android. Disponible en:
http://developer.android.com/reference/packages.html

Documentación sobre OpenCv. Disponible en:
http://docs.opencv.org/modules/refman.html

Descripción del Android Manifest. Disponible en:
http://developer.android.com/guide/topics/manifest/manifest-intro.html

Descripción del Application.mk. Disponible en:
http://www.kandroid.org/ndk/docs/APPLICATION-MK.html

Descripción del Android.mk. Disponible en:
http://www.kandroid.org/ndk/docs/ANDROID-MK.html

Qué son los layouts, cómo funcionan y qué tipos hay. Disponible en:
http://www.sgoliver.net/blog/?p=1341

Sobre las SharedPreferences. Disponible en:
http://www.sgoliver.net/blog/?p=1731

Tutorial de JNI. Disponible en:
https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaNativeInterface.html#zz-3.

Tutorial de JNI + OpenCv Android. Disponible en:
http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/android_binary_package_using_with_NDK.html

Tutorial para activar el modo depuración. Disponible en:
http://www.fandroides.com/que-es-y-como-habilitar-la-depuracion-usb-en-android/

Tutorial para instalar aplicaciones desde un apk en el PC. Disponible en:
http://www.talkandroid.com/guides/beginner/install-apk-files-on-android/

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Wed Jan 07 22:03:01 CET 2015
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)